Conditional functional dependencies for capturing data inconsistencies

Conditional Functional Dependencies for Capturing Data Inconsistencies

WENFEI FAN
University of Edinburgh & Bell Laboratories
and
FLORIS GEERTS and XIBEI JIA
University of Edinburgh
and
ANASTASIOS KEMENTSIETSIDIS
IBM Watson

We propose a class of integrity constraints for relational databases, referred to as *conditional functional dependencies* (CFDs), and study their applications in data cleaning. In contrast to traditional functional dependencies (FDs) that were developed mainly for schema design, CFDs aim at capturing the consistency of data by enforcing bindings of semantically related values. For static analysis of CFDs we investigate *the consistency problem*, which is to determine whether or not there exists a nonempty database satisfying a given set of CFDs, and *the implication problem*, which is to decide whether or not a set of CFDs entails another CFD. We show that while any set of transitional FDs is trivially consistent, the consistency problem is NP-complete for CFDs, but it is in PTIME when either the database schema is predefined or no attributes involved in the CFDS have a finite domain. For the implication analysis of CFDs, we provide an inference system analogous to Armstrong's axioms for FDs, and show that the implication problem is coNP-complete for CFDs in contrast to the linear-time complexity for their traditional counterpart. We also present an algorithm for computing a minimal cover of a set of CFDs. Since CFDs allow data bindings, in some cases CFDs may be physically large, complicating detection of constraint violations. We develop techniques for detecting CFD violations in SQL as well as novel techniques for checking multiple constraints in a single query. We also provide incremental methods for checking CFDs in response to changes to the database. We experimentally verify the effectiveness of our CFD-based methods for inconsistency detection. This work not only yields a constraint theory for CFDs but is also a step toward a practical constraint-based method for improving data quality.

Categories and Subject Descriptors: H.2.m [**Database Management**]: Miscellaneous—*Data cleaning*; H.2.1 [**Database Management**]: Logical Design—*Schema and subschema*

General Terms: Algorithms, Experimentation, Performance, Theory

Additional Key Words and Phrases: Data cleaning, Functional dependency, SQL

1. INTRODUCTION

Recent statistics reveal that dirty data costs US businesses billions of dollars annually (cf. [Eckerson 2002]). It is also estimated that data cleaning, a labor-intensive and complex process, accounts for 30%-80% of the development time in a typical data warehouse project (cf. [Shilakes and Tylman 1998]). These highlight the need for data-cleaning tools to automatically detect and effectively remove inconsistencies and errors in the data.

One of the most important questions in connection with data cleaning is how to model the consistency of the data, *i.e.,* how to specify and determine that the data is clean? This calls for appropriate application-specific integrity constraints [Rahm and Do 2000] to model the fundamental semantics of the data. However, little previous work has studied this issue. Commercial ETL (extraction, transformation, loading) tools have little built-in data cleaning capability, and a significant portion of the cleaning work has still to be done manually or by low-level programs that are difficult to write and maintain [Rahm and Do 2000]. A bulk of prior research has focused on the merge-purge problem for the elimination of *approximate duplicates* (*e.g.,* [Winkler 1994; Hernandez and Stolfo 1998; Galhardas et al. 2000; Monge 2000]), or on detecting domain discrepancies and structural conflicts (*e.g.,* [Raman and Hellerstein 2001]).

There has also been recent work on *constraint repair* [Arenas et al. 2003; Franconi et al. 2001; Bravo and Bertossi 2003; Cali et al. 2003a; 2003b; Chomicki and Marcinkowski 2005a; Greco et al. 2003; Wijsen 2005], which specifies the consistency of data in terms of constraints, and detects inconsistencies in the data as violations of the constraints. However, previous work on constraint repair is mostly based on traditional dependencies (*e.g.,* functional and full dependencies, etc), which were developed mainly for schema design, but are often insufficient to capture the semantics of the data, as illustrated by the example below.

Example 1.1. Consider a relation schema $\mathsf{cust}(\mathsf{CC}, \mathsf{AC}, \mathsf{PN}, \mathsf{NM}, \mathsf{STR}, \mathsf{CT}, \mathsf{ZIP})$, which specifies a customer in terms of the customer's phone (country code (CC), area code (AC), phone number (PN)), name (NM), and address (street (STR), city (CT), zip code (ZIP)). An instance of cust is shown in Fig. 1.

Traditional functional dependencies (FDs) on a cust relation may include:

f_1: $[\mathsf{CC}, \mathsf{AC}, \mathsf{PN}] \rightarrow [\mathsf{STR}, \mathsf{CT}, \mathsf{ZIP}]$
f_2: $[\mathsf{CC}, \mathsf{AC}] \rightarrow [\mathsf{CT}]$

Recall the semantics of an FD: f_1 requires that customer records with the same country code, area code and phone number also have the same street, city and zip code. Similarly, f_2 requires that two customer records with the same country- and area codes also have the same city name. Traditional FDs are to hold on all the tuples in the relation (indeed they do on Fig. 1).

In contrast, the following constraint is supposed to hold only when the country code is 44. That is, for customers in the UK, ZIP determines STR:

ϕ_0: $[\mathsf{CC} = 44, \mathsf{ZIP}] \rightarrow [\mathsf{STR}]$

In other words, ϕ_0 is an FD that is to hold on the subset of tuples that satisfies the pattern "$\mathsf{CC} = 44$", rather than on the entire cust relation. It is generally *not*

	CC	AC	PN	NM	STR	CT	ZIP
t_1:	01	908	1111111	Mike	Tree Ave.	NYC	07974
t_2:	01	908	1111111	Rick	Tree Ave.	NYC	07974
t_3:	01	212	2222222	Joe	Elm Str.	NYC	01202
t_4:	01	212	2222222	Jim	Elm Str.	NYC	02404
t_5:	01	215	3333333	Ben	Oak Ave.	PHI	02394
t_6:	44	131	4444444	Ian	High St.	EDI	EH4 1DT

Fig. 1. An instance of the cust relation

considered an FD in the standard definition since ϕ_0 includes a *pattern* with *data values* in its specification.

The following constraints are again not considered FDs:

ϕ_1: $[\mathsf{CC} = 01, \mathsf{AC} = 908, \mathsf{PN}] \rightarrow [\mathsf{STR}, \mathsf{CT} = \text{MH}, \mathsf{ZIP}]$
ϕ_2: $[\mathsf{CC} = 01, \mathsf{AC} = 212, \mathsf{PN}] \rightarrow [\mathsf{STR}, \mathsf{CT} = \text{NYC}, \mathsf{ZIP}]$
ϕ_3: $[\mathsf{CC} = 01, \mathsf{AC} = 215] \rightarrow [\mathsf{CT} = \text{PHI}]$

Constraint ϕ_1 assures that only in the US (country code 01) and for area code 908, if two tuples have the same PN, then they must have the same STR and ZIP and moreover, the city *must* be MH. Similarly, ϕ_2 assures that if the area code is 212 then the city must be NYC; and ϕ_3 specifies that for all tuples in the US and with area code 215, their city must be PHI (irrespective of the values of the other attributes). Observe that ϕ_1 and ϕ_2 *refine* the standard FD f_1 given above, while ϕ_3 refines the FD f_2. These refinements essentially enforce bindings of semantically related data values. Indeed, while tuples t_1 and t_2 in Fig. 1 do not violate f_1, they violate its refinement ϕ_1, since the city cannot be NYC if the area code is 908.

In this example, the constraints ϕ_0, ϕ_1, ϕ_2 and ϕ_3 capture a fundamental part of the semantics of the data. However, they cannot be expressed as standard FDs. Indeed, in contrast to FDs that express knowledge only at the intensional (schema) level, these constraints combine intensional and extensional (data-level) expressions by incorporating *constants* into FDs and enforcing patterns of semantically related data values. Further, they hold *conditionally*, *i.e.,* only on the subset of a relation that satisfies certain patterns, rather than on the entire relation.

Constraints that hold conditionally may arise in a number of domains. For example, an employee's title may determine her pay grade in some parts of an organization but not in others; an individual's address may determine his tax rate in some countries while in others it may depend on his salary, etc. Further, dependencies that apply conditionally appear to be particularly needed when integrating data, since dependencies that hold only in a subset of sources will hold only conditionally in the integrated data. Indeed, in the authors' direct experience with telecommunication service providers, errors and inconsistencies commonly found in real-life data often emerge as violations of conditional dependencies; they routinely lead to problems like failure to bill for provisioned services, delay in repairing network problems, unnecessary leasing of equipment, and so on, but cannot be detected by schema-level constraints alone. These call for the study of conditional dependencies, which aim to capture data inconsistencies at the intentional level (as done in prior work on data cleaning) and extensional level in a *uniform framework*. A recent study [Cong et al. 2007] also demonstrates that conditional dependencies are

more effective than standard FDs in repairing inconsistent data in practice.

There has been work on extending traditional equality-generating and tuple-generating dependencies (EGDs and TGDs, which subsume functional and inclusion dependencies, respectively) by incorporating constants, *i.e.,* combining intensional and extensional expressions [Bra and Paredaens 1983; Maher and Srivastava 1996; Maher 1997; Baudinet et al. 1999]. These extensions were proposed for constraint logic programming languages and constraint databases, but have *not* been considered in previous work on data cleaning. As will be seen in Section 7, some of these extensions cannot express ϕ_0–ϕ_3 given above, and those that can express conditional dependencies incur a higher complexity. For data cleaning, one typically needs a simple extension of traditional FDs that suffices to capture data inconsistencies commonly found in practice, without incurring unnecessary complexity.

In response to the practical need, this paper introduces a novel extension of traditional FDs, referred to as *conditional functional dependencies* (CFDs), that are capable of capturing the notion of "correct data" in these situations. A CFD extends an FD by incorporating a pattern tableau that enforces binding of semantically related values. Unlike its traditional counterpart, the CFD is required to hold only on tuples that satisfy a pattern in the pattern tableau, rather than on the entire relation. For example, all the constraints we have encountered so far can be expressed as CFDs. A formal framework for modeling CFDs is our first contribution.

Since CFDs are an extension of standard FDs, it is natural to ask whether or not we can still effectively reason about CFDs along the same lines as their FD counterpart. Does a set Σ of CFDs make sense, *i.e.,* are the CFDs consistent (is there a nonempty database that satisfies Σ)? Is there an inference system, analogous to Armstrong's Axioms for FDs, to effectively determine whether or not a set of CFDs implies (entails) another CFD? These are the classical consistency and implication problems typically associated with integrity constraints. These problems are not only fundamental to CFDs, but are also important for data cleaning. Indeed, if an input set Σ of CFDs is found inconsistent, then there is *no need* to check (validate) the CFDs against the data at all. Further, it helps the user discover errors in CFD specification. When Σ is consistent, an effective implication analysis would allow us to find a *minimal cover* Σ_{mc} of Σ that is equivalent to Σ but contains no redundant CFDs, patterns or attributes; it is typically more efficient to use Σ_{mc} instead of Σ when detecting and removing inconsistencies from the data.

Our second contribution consists of techniques for the consistency analysis of CFDs. We show that the static analysis of CFDs introduces new challenges. Indeed, a set of CFDs may be inconsistent, *i.e.,* there may not exist a nonempty database satisfying the CFDs, a problem not encountered when dealing with traditional FDs. We show that the consistency problem for CFDs is NP-complete in general, but it is in PTIME when either the database schema is predefined or no attributes involved in the CFDs have a finite domain. To cope with the intractability we provide an approximation factor preserving reduction to the well-studied MAXGSAT problem. This allows us to leverage existing approximation algorithms for MAXGSAT to check the consistency of CFDs (see, *e.g.,* [Papadimitriou 1994; Vazirani 2003] for approximation factor preserving reductions and approximation algorithms for MAXGSAT).

Our third contribution is a sound and complete inference system for the implica-

tion analysis of CFDs, which is analogous to but is more involved than Armstrong's Axioms for FDs (see, *e.g.,* [Abiteboul et al. 1995]). We show that as opposed to standard FDs for which the implication problem is in linear-time, the implication analysis for CFDs is coNP-complete. We also identify a special case when the implication problem is in PTIME. Based on these we present a technique for computing a minimal cover of a set of CFDs. These results are not only useful for data cleaning as an optimization technique by minimizing the input CFDs, but also yield a CFD theory analogous to the theory of FDs.

Our fourth contribution is the development of SQL techniques for detecting CFD violations. Since CFDs incorporate data values, they may in some cases be physically large, and straightforward techniques may lead to a very large number of detection queries. We develop nontrivial techniques to *merge* and efficiently check *a set* of CFDs even with a very large number of conditions. These guarantee: (a) a *single* pair of SQL queries is generated, with a bounded size independent of the pattern tableaux in the CFDs, and (b) only two passes of the database are needed. Furthermore, we develop techniques to incrementally detect CFD violations, as the underlying instance changes due to the insertions or deletions of tuples.

Our fifth and final contribution is an experimental study of the performance of our detection techniques as data size and constraint complexity vary. We find that our techniques allow violations of CFDs with even a large number of conditions to be detected efficiently on large data sets. However, we also find that care must be taken to present the complicated **where** clauses generated by our technique to the optimizer in a way that can be easily optimized. We illustrate that certain rewritings of our detection queries are more prone to optimizations than others, by comparing their performance under various settings. More importantly, in situations where the optimizer is unable to cope with the complexity of the detection queries, we offer alternative relational representations of our solutions, and we illustrate that these alternatives result in detection queries that are simpler, easier to optimize, and faster to execute. In addition, we demonstrate that our incremental techniques vastly outperform, in terms of time, batch algorithms that detect violations starting from scratch in response to changes to the underlying database.

Our conclusion is that CFDs are a promising tool for improving data quality. While a facility to detect inconsistencies emerged as CFD violations will logically become part of the cleaning process supported by data cleaning systems, we are not aware of analogous functionality in any of the existing systems.

Organization. The remainder of the paper is organized as follows. Section 2 formally defines CFDs. This is followed by the static analysis of CFDs: Section 3 studies the consistency analysis of CFDs, and Section 4 provides the inference system for CFD implication. Section 5 presents our SQL techniques for detecting and incrementally detecting CFD violations, followed by the experimental study in Section 6. Section 7 discusses related work, followed by topics for future work in Section 8.

2. CONDITIONAL FUNCTIONAL DEPENDENCIES

In this section we define conditional functional dependencies (CFDs). Consider a relation schema R defined over a fixed set of attributes, denoted by $\mathsf{attr}(R)$. For each attribute $A \in \mathsf{attr}(R)$, its domain is specified in R, denoted as $\mathsf{dom}(A)$.

(a) Tableau T_1 of $\varphi_1 = ([\mathsf{CC}, \mathsf{ZIP}] \rightarrow [\mathsf{STR}],\ T_1)$

CC	ZIP	STR
44	_	_

(b) Tableau T_2 of $\varphi_2 = ([\mathsf{CC}, \mathsf{AC}, \mathsf{PN}] \rightarrow [\mathsf{STR}, \mathsf{CT}, \mathsf{ZIP}],\ T_2)$

CC	AC	PN	STR	CT	ZIP
_	_	_	_	_	_
01	908	_	_	MH	_
01	212	_	_	NYC	_

(c) Tableau T_3 of $\varphi_3 = ([\mathsf{CC}, \mathsf{AC}] \rightarrow [\mathsf{CT}],\ T_3)$

CC	AC	CT
_	_	_
01	215	PHI
44	141	GLA

Fig. 2. Example CFDs

Syntax. A CFD φ on R is a pair $(R : X \rightarrow Y,\ T_p)$, where (1) X, Y are sets of attributes in $\mathsf{attr}(R)$, (2) $X \rightarrow Y$ is a standard FD, referred to as the FD *embedded in* φ; and (3) T_p is a tableau with attributes in X and Y, referred to as the *pattern tableau* of φ, where for each A in $X \cup Y$ and each tuple $t \in T_p$, $t[A]$ is either a constant 'a' in $\mathsf{dom}(A)$, or an unnamed variable '_' that draws values from $\mathsf{dom}(A)$.

If A occurs in both X and Y, we use $t[A_L]$ and $t[A_R]$ to indicate the occurrence of A in X and Y, respectively, and separate the X and Y attributes in a pattern tuple with '$\|$'. We write φ as $(X \rightarrow Y,\ T_p)$ when R is clear from the context, and denote X as $\text{LHS}(\varphi)$ and Y as $\text{RHS}(\varphi)$.

Example 2.1. The constraints $\phi_0, f_1, \phi_1, \phi_2, f_2, \phi_3$ on the cust table given in Example 1.1 can be expressed as CFDs φ_1 (for ϕ_0), φ_2 (for f_1, ϕ_1 and ϕ_2, one per line, respectively) and φ_3 (for f_2, ϕ_3 and an additional $[\mathsf{CC} = 44, \mathsf{AC} = 141] \rightarrow [\mathsf{CT} = \text{GLA}]$ to be used in Section 5), as shown in Fig. 2.

If we represent both data and constraints in a uniform tableau format, then at one end of the spectrum are relational tables which consist of data values without logic variables, and at the other end are traditional constraints which are defined in terms of logic variables but without data values, while CFDs are in the between.

Semantics. For a pattern tuple t_p in T_p, we define an *instantiation* ρ to be a mapping from t_p to a *data tuple* with no variables, such that for each attribute A in $X \cup Y$, if $t_p[A]$ is '_', ρ maps $t_p[A]$ to a *constant* in $\mathsf{dom}(A)$, and if $t_p[A]$ is a constant 'a', ρ maps $t_p[A]$ to the *same* value 'a'. For example, for $t_p[A, B] = (a, \_)$, one can define an instantiation ρ such that $\rho(t_p[A, B]) = (a, b)$, which maps $t_p[A]$ to itself and $t_p[B]$ to a value 'b' in $\mathsf{dom}(B)$. Obviously, for an attribute A occurring in both X and Y, we require that $\rho(t_p[A_L]) = \rho(t_p[A_R])$. Note that an instantiation ρ may map different occurrences of '_' in t_p to different constants; *e.g.,* if $t_p[A, B] = (\_, \_)$, then $\rho(t_p[A, B]) = (a, b)$ is well-defined if $a \in \mathsf{dom}(A)$ and $b \in \mathsf{dom}(B)$.

A data tuple t is said to *match* a pattern tuple t_p, denoted by $t \asymp t_p$, if there is an instantiation ρ such that $\rho(t_p) = t$. For example, $t[A, B] = (a, b) \asymp t_p[A, B] = (a, \_)$.

An instance I of R *satisfies* the CFD φ, denoted by $I \models \varphi$, if for *each pair* of tuples t_1, t_2 in the instance I, and for *each* tuple t_p in the pattern tableau T_p of φ,

if $t_1[X] = t_2[X] \asymp t_p[X]$, then $t_1[Y] = t_2[Y] \asymp t_p[Y]$. That is, if $t_1[X]$ and $t_2[X]$ are equal and in addition, they both match the pattern $t_p[X]$, then $t_1[Y]$ and $t_2[Y]$ must also be equal to each other and both match the pattern $t_p[Y]$.

Intuitively, each tuple t_p in the pattern tableau T_p of φ is a constraint defined on the set $I_{(\varphi,t_p)} = \{t \mid t \in I, t[X] \asymp t_p[X]\}$ such that for any $t_1, t_2 \in I_{(\varphi,t_p)}$, if $t_1[X] = t_2[X]$, then (a) $t_1[Y] = t_2[Y]$, and (b) $t_1[Y] \asymp t_p[Y]$. Here (a) enforces the semantics of the embedded FD, and (b) assures the binding between *constants* in $t_p[Y]$ and *constants* in $t_1[Y]$. Note that this constraint is defined on the subset $I_{(\varphi,t_p)}$ of I identified by $t_p[X]$, rather than on the entire instance I.

If Σ is a set of CFDs, we write $I \models \Sigma$ if $I \models \varphi$ for *each* CFD $\varphi \in \Sigma$. If a relation $I \models \Sigma$, then we say that I is *clean* with respect to Σ.

Example 2.2. The cust relation in Fig. 1 satisfies φ_1 and φ_3 of Fig. 2. However, it does not satisfy φ_2. Indeed, tuple t_1 *violates* the pattern tuple $t_p = (01, 908, \_ \parallel \_,$ MH, $\_)$ in tableau T_2 of φ_2: $t_1[\mathsf{CC}, \mathsf{AC}, \mathsf{PN}] = t_1[\mathsf{CC}, \mathsf{AC}, \mathsf{PN}] \asymp (01, 908, \_)$, but $t_1[\mathsf{STR}, \mathsf{CT}, \mathsf{ZIP}] \not\asymp (\_,$ MH, $\_)$ since $t_1[\mathsf{CT}]$ is NYC instead of MH; similarly for t_2.

This example tells us that while violations of standard FDs require *two* tuples, a *single* tuple may violate a CFD.

Observe that a standard FD $X \to Y$ can be expressed as a CFD $(X \to Y,\ T_p)$ in which T_p contains a single tuple consisting of '_' only, without constants. For example, if we let T_3 of φ_3 in Fig. 2 contain only $(\_, \_ \parallel \_)$, then it is the CFD representation of the FD f_2 given in Example 1.1.

To conclude this section we introduce a normal from for CFDs. A CFD φ is *in the normal form* if $\varphi = (R : X \to A, \{t_p\})$, written as $(R : X \to A, t_p)$, where A is a single attribute and the pattern tableau consists of a single pattern tuple t_p only.

We say that two sets Σ_1 and Σ_2 of CFDs are *equivalent*, denoted by $\Sigma_1 \equiv \Sigma_2$, if for any instance I of R, $I \models \Sigma_1$ if and only if $I \models \Sigma_2$.

PROPOSITION 2.3. *For any set Σ of CFDs, there exists a set Σ_{nf} of CFDs such that each φ in Σ_{nf} is in the normal form, and $\Sigma \equiv \Sigma_{nf}$. Moreover, Σ_{nf} consists of at most $|\mathsf{attr}(R)|\|\Sigma\|$ many CFDs, where $\|\Sigma\|$ denotes the total number of pattern tuples in CFDs in Σ.*

PROOF. Given Σ we compute $\Sigma_{\rm nf}$ as follows. First, for each CFD $\varphi = (R : X \to Y, T_p)$ in Σ, we associate with φ a set Σ_φ consisting of $|T_p|$ many CFDs of the form $(R : X \to Y, t_p)$ for each $t_p \in T_p$. Next, for each CFD $\psi \in \Sigma_\varphi$, assuming *w.l.o.g.* that $\psi = (R : Y \to [B_1, \ldots, B_k], t_p)$, we define $\psi_{B_i} = (R : Y \to B_i, t_p[Y \cup \{B_i\}])$ for each $i \in [1, k]$. Define $\Sigma_{\rm nf}$ to be $\bigcup_{\varphi\in\Sigma} \bigcup_{\psi\in\Sigma_\varphi, B_i\in{\rm RHS}(\psi)} \psi_B$, which is a set of CFDs in the normal form. From the semantics of CFDs it is clear that $\{\psi\} \equiv \bigcup_{B_i\in{\rm RHS}(\psi)} \psi_{B_i}$, $\{\varphi\} \equiv \Sigma_\varphi$, $\Sigma \equiv \bigcup_{\varphi\in\Sigma} \Sigma_\varphi$, and hence $\Sigma \equiv \Sigma_{\rm nf}$. □

In the sequel we consider CFDs in the normal form, unless stated otherwise.

3. CONSISTENCY ANALYSIS OF CONDITIONAL FUNCTIONAL DEPENDENCIES

We now investigate classical decision problems associated with CFDs. We focus on consistency analysis in this section, and study implication analysis in Section 4.

3.1 Reasoning about the Consistency of Conditional Functional Dependencies

One can specify any set of standard FDs, without worrying about consistency. This is no longer the case for CFDs, as illustrated by the example below.

Example 3.1. Consider CFD $\psi_1 = (R : [A] \to [B], T_1)$, where T_1 consists of two pattern tuples $(\_ \parallel b)$ and $(\_ \parallel c)$. No nonempty instance I of R can possibly satisfy ψ_1. Indeed, for any tuple t in I, while the first pattern tuple says that $t[B]$ must be b no matter what value $t[A]$ has, the second pattern requires $t[B]$ to be c.

Now assume that $\mathsf{dom}(A)$ is bool. Consider two CFDs $\psi_2 = (R : [A] \to [B], T_2)$ and $\psi_3 = (R : [B] \to [A], T_3)$, where T_2 has two patterns $(\mathsf{true} \parallel b_1)$, $(\mathsf{false} \parallel b_2)$, and T_3 contains $(b_1 \parallel \mathsf{false})$ and $(b_2 \parallel \mathsf{true})$. While ψ_2 and ψ_3 can be separately satisfied by a nonempty instance, there exists no nonempty instance I such that $I \models \{\psi_2, \psi_3\}$. Indeed, for any tuple t in I, no matter what Boolean value $t[A]$ has, ψ_2 and ψ_3 together force $t[A]$ to take the other value from the finite domain bool. This tells us that attributes with a finite domain may complicate the consistency analysis. Note that if $\mathsf{dom}(A)$ contains one extra value, say maybe, then the instance $I = \{(\mathsf{maybe}, b_3)\}$, for b_3 distinct from both b_1 and b_2, satisfies both ψ_2 and ψ_3.

Consistency. A set Σ of CFDs on a schema R is said to be *consistent* if there exists a nonempty instance I of R such that $I \models \Sigma$. The *consistency problem for* CFD*s* is to determine, given a set Σ of CFDs on a schema R, whether or not Σ is consistent.

Intuitively, the consistency analysis is to determine whether a given set Σ makes sense or not. One might be tempted to adopt a stronger notion of consistency, as suggested by the following example. Consider a CFD $\psi_4 = (R : [A] \to [B], T_4)$, where T_4 consists of pattern tuples $(a \parallel b)$ and $(a \parallel c)$ and $b \neq c$. There is obviously a nonempty instance I of R that satisfy ψ_4. However, the conflict between the pattern tuples in ψ_4 becomes evident if I contains a tuple t with $t[A] = a$. Indeed, the first pattern tuple forces $t[B]$ to be b while the second pattern tuple requires $t[B]$ to be c. In light of this one might want to ensure that every CFD in Σ does not conflict with the rest of Σ no matter over what instances of R. This can be guaranteed by requiring for each CFD $\varphi = (R : X \to A, (t_p \parallel a)) \in \Sigma$ the existence of an instance $I_\varphi \models \Sigma$ such that I_φ contains a tuple t with $t[X] \asymp t_p$. However, this stronger notion of consistency is equivalent to the "classical" notion of consistency. To see this, we first introduce the notions of total CFDs and constant attributes.

A *total* CFD is of the form $(R : X \to A, (t_p \parallel a))$, where t_p consists of '$\_$' only. It assures that all tuples in a relation must take the same value 'a' in the A attribute.

The *constant attributes* of a CFD $\varphi = (R : X \to A, (t_p \parallel a))$ is the set $X_{(c,\varphi)}$ of all those attributes in X such that for any $B \in X_{(c,\varphi)}$, $t_p[B]$ is a constant.

One can check the strong consistency of a set Σ of CFDs as follows. For each $\phi \in \Sigma$, define a set $\Sigma \cup \{(R : [C] \to [C], (\_ \parallel t_p[C])) \mid C \in X_{(c,\phi)}\}$. It is easy to verify that Σ is strongly consistent iff all these sets are consistent. For example, for $\Sigma = \{\psi_4\}$, since $\Sigma \cup \{(R : [A] \to [A], (\_ \parallel a))\}$ is not consistent it follows that Σ is not strongly consistent. It is thus sufficient to consider the consistency of CFDs.

Intractability. As opposed to the fact that any set of standard FDs is consistent, the consistency analysis of CFDs is nontrivial, as shown below.

THEOREM 3.2. *The consistency problem for* CFD*s is* NP*-complete.*

PROOF. We first show that the problem is in NP. Consider a set Σ of CFDs defined on a schema R. The consistency problem has the following *small model property*: if there exists a nonempty instance I of R such that $I \models \Sigma$, then for any tuple $t \in I$, $I_t = \{t\}$ is an instance of R and $I_t \models \Sigma$. Thus it suffices to consider single-tuple instances $I = \{t\}$ for deciding whether Σ is consistent. Assume *w.l.o.g.* that $\mathsf{attr}(R) = \{A_1, \ldots, A_n\}$. Moreover, for each $i \in [1, n]$, let the active domain $\mathsf{adom}(A_i)$ of A consist of all constants of $t_p[A_i]$ for all pattern tuples t_p in Σ, plus at most an extra distinct value from $\mathsf{dom}(A_i)$; then it is easy to verify that Σ is consistent iff there exists a mapping ν from $t[A_i]$ to $\mathsf{adom}(A_i)$ such that $I' = \{(\nu(t[A_1]), \ldots, \nu(t[A_n]))\}$ and $I' \models \Sigma$. Based on these, we give a NP algorithm for checking the consistency of Σ as follows: (a) Guess a single tuple t of R such that $t[A_i] \in \mathsf{adom}(A_i)$. (b) Check whether or not $I = \{t\}$ satisfies Σ. Obviously step (b) can be done in PTIME in the size $|\Sigma|$ of Σ, and thus this algorithm is in NP.

We next show that the problem is NP-hard by reduction from the non-tautology problem. An instance of the non-tautology problem is a well-formed Boolean formula $\phi = C_1 \vee \cdots \vee C_n$, where all the variables in ϕ are $x_1, \ldots, x_m$, C_j is of the form $\ell_{j_1} \wedge \ell_{j_2} \wedge \ell_{j_3}$, and ℓ_{i_j} is either x_s or $\bar{x}_s$, for $s \in [1, m]$. The problem is to determine whether there is a truth assignment such that ϕ is false, *i.e.,* ϕ is not valid. This problem is known to be NP-complete (cf. [Garey and Johnson 1979]).

Given an instance ϕ of the non-tautology problem, we define an instance of the consistency problem for CFDs, namely, a relation schema R and a set Σ of CFDs on R, such that ϕ is not a tautology if and only if Σ is consistent. We define R to be $(X_1, \ldots, X_m, C)$, in which all the attributes are Boolean. Intuitively, for each tuple t in an instance I of R, $t[X_1, \ldots, X_m]$ encodes a truth assignment of the variables $x_1, \ldots, x_m$. We next define a set Σ of CFDs, and illustrate the C attribute.

(a) $(R : [X_1, \ldots, X_m] \to C, T_p)$, where for each clause C_j, for $j \in [1, n]$, T_p contains a tuple t_j such that $t_j[C] = 1$, and moreover, for each $i \in [1, m]$, $t_j[X_i] = 1$ if x_i appears in C_j, $t_j[X_i] = 0$ if $\bar{x}_i$ appears in C_j, and $t_j[X_i] = \_$ otherwise. Hence, any tuple t in an instance I of R that satisfies this CFD must have $t[C] = 1$ in case the truth assignment corresponding to t makes at least one of the clause in ϕ true.

(b) $(R : C \to C, \{(1 \parallel 0)\})$. This is to assure that none of the clauses is satisfied, *i.e.,* ϕ is not a tautology. Indeed, this CFD prevents any tuple t (*i.e.,* truth assignment) in an instance I of R that satisfies this CFD from having $t[C] = 1$.

We now show the correctness of this reduction. Clearly, any instance I of R that satisfies both CFDs provides us with truth assignments of the variables in ϕ that make ϕ false. Thus, ϕ is not a tautology. Conversely, if ϕ is not a tautology, then there is a truth assignment ν such that ϕ is false. We construct an instance I of R consisting of a single tuple t such that $t[C] = 0$, and $t[X_i] = 1$ if $\nu(x_i) = 1$ and $t[X_i] = 0$ otherwise. It is easy to verify that I satisfies Σ.

Putting these together, we have that the consistency problem is NP-complete. □

Tractable cases and a consistency checking algorithm. In light of the intractability, we identify several tractable special cases of the consistency problem. We first present two subclasses of CFDs which are always consistent (Propositions 3.3 and 3.4). These simple syntactic restrictions on CFDs provide sufficient conditions for retaining the trivial consistency analysis of their standard counterpart. We

then show, by giving a consistency checking algorithm, that when all attributes involved in a set of CFDs have an infinite domain, the consistency analysis is in PTIME (Proposition 3.5). These reveal certain insight of the intractability: the presence of finite-domain attributes complicates the consistency analysis. Finally, we extend the checking algorithm to arbitrary CFDs, with finite-domain attributes or not, and identify a PTIME case in the presence of finite-domain attributes (Proposition 3.6).

We first observe that inconsistencies can only be caused by the presence of CFDs that have a constant in their RHS. Indeed, let Σ be a set of CFDs and denote by Σ_c the set of CFDs in Σ of the form $(R : X \to A, (t_p \parallel a))$ for some constant $a \in \mathsf{dom}(A)$. Then Σ is consistent iff Σ_c is consistent. Indeed, the consistency of Σ implies the consistency of Σ_c. For the other direction, let I be an instance of R such that $I \models \Sigma_c$. Select a single tuple $t \in I$ and define $J = \{t\}$. Clearly, $J \models \Sigma_c$. Let Σ_v be the subset of CFDs in Σ of the form $(R : Y \to B, (s_p \parallel \_))$. It suffices to observe that any $\psi \in \Sigma_v$ is trivially satisfied on single-tuple instances. Hence, $J \models \Sigma_v$ and therefore $J \models \Sigma$. From this it follows:

PROPOSITION 3.3. *For any set Σ of* CFDs *that contain no* CFDs *with a constant* RHS*, Σ is always consistent.*

Due to this in the sequel we only consider Σ that solely consists of CFDs with a constant RHS, *i.e.,* CFDs of the form $(R : X \to A, (t_p \parallel a))$ with $a \in \mathsf{dom}(A)$.

Let $\mathsf{attr}(\Sigma)$ be the set of attributes in $\mathsf{attr}(R)$ that appear in some CFDs in Σ. Clearly $|\mathsf{attr}(\Sigma)| \leq |\mathsf{attr}(R)|$. We first consider the setting in which $\mathsf{attr}(\Sigma)$ does not contain any finite domain attributes and Σ does not contain any total CFDs.

PROPOSITION 3.4. *For any set Σ of* CFDs *defined on a schema R, if no* CFD *in Σ is total and all attributes in $\mathsf{attr}(\Sigma)$ have an infinite domain, then Σ is consistent.*

PROOF. By the small property observed in the proof of Theorem 3.2, it suffices to consider single-tuple relations when checking the consistency of Σ. For any instance $I = \{t\}$ of R, if for each attribute $A \in \mathsf{attr}(R)$, $t[A]$ is a constant not appearing in any pattern tuple in Σ, then $I \models \Sigma$. Indeed, any CFD in Σ is of the form $\varphi = (R : X \to A, (t_p \parallel a))$ with t_p a pattern tuple that contains at least one constant, since Σ does not contain total CFDs. Thus by the choice of constants in t, I does not match the $\text{LHS}(\varphi)$ and hence trivially satisfies φ, for any φ in Σ. □

The result below shows that the presence of total CFDs makes our lives harder.

PROPOSITION 3.5. *For any set Σ of* CFDs *defined on a schema R (possibly containing total* CFDs*), the consistency of Σ can be determined in $O(|\Sigma|^2|\mathsf{attr}(\Sigma)|)$ time if all attributes in $\mathsf{attr}(\Sigma)$ have an infinite domain.*

PROOF. Again due to the small model property given in the proof of Theorem 3.2, it suffices to consider single-tuple instances $I = \{t\}$ of R when checking the consistency of Σ. In contrast to the proof of Proposition 3.4, the presence of total CFDs in Σ does not allow us to choose the constants in t arbitrarily. Moreover, the presence of certain constants in some attributes of t might induce constants in other attributes as enforced by the CFDs in Σ. The key observation for consistency checking is that Σ is consistent if and only if there exists no attribute A in $\mathsf{attr}(R)$ for which two *different* constants for $t[A]$ are enforced by Σ.

We now give an algorithm, called CONSISTENCY, that checks whether this situation arises. The algorithm starts with a set $\mathcal{S}_0$ of the attribute-value pairs found in the total CFDs in Σ, *i.e.,* $\mathcal{S}_0 = \{(A, a) \mid \varphi = (R : X \to A, (\_, \ldots, \_ \parallel a)),\ \varphi \in \Sigma\}$. Then, at step i, the algorithm expands the current set $\mathcal{S}_{i-1}$ of attribute-value pairs to $\mathcal{S}_i$ as follows: for any $\phi = (R : X \to B, (t_p \parallel b)) \in \Sigma$, if for each $A_j \in X_{(c,\phi)}$, $(A_j, t_p[A_j])$ is already in $\mathcal{S}_{i-1}$, then add (B, b) to $\mathcal{S}_i$, where $X_{(c,\phi)}$ is the set of constant attributes of ϕ. After $\mathcal{S}_i$ is constructed, CONSISTENCY checks whether there exist two pairs (A, a_1) and (A, a_2) in $\mathcal{S}_i$ with $a_1 \neq a_2$. If so, the algorithm stops and concludes that Σ is inconsistent. Otherwise, it proceeds with step $(i + 1)$. If after some step $\mathcal{S}_{i+1} = \mathcal{S}_i$, then the algorithm stops and concludes that Σ is consistent.

The worst-case time complexity of the algorithm is $O(|\Sigma|^2|\mathsf{attr}(\Sigma)|)$. Indeed, the number of steps is bounded by $|\Sigma|$, since in each step at least one CFD in Σ can be used to induce some new attribute-value pair. Moreover, at each step $|\Sigma|$ many CFDs are checked. Further, checking whether a CFD induces an attribute-value pair takes at most $O(|\mathsf{attr}(\Sigma)|)$ time. From this follows the complexity.

To show the correctness of algorithm CONSISTENCY, first assume that the algorithm concludes that Σ is consistent. Denote by $\mathcal{S}$ the final set of attribute-value pairs computed by CONSISTENCY. Then we define $I = \{t\}$ with $t[A_i] = a_i$ for $(A_i, a_i) \in \mathcal{S}$, and for all other attributes B we set $t[B]$ to a constant that does not appear in any pattern tuple of CFDs in Σ (this is doable since $\mathsf{attr}(\Sigma)$ contains infinite-domain attributes only). We show that $I \models \Sigma$, *i.e.,* Σ is indeed consistent. Assume by contradiction that there exists a CFD $\varphi = (R : X \to A, (t_p \parallel a)) \in \Sigma$ such that $I \not\models \varphi$. This necessarily implies that $(A_i, t_p[A_i]) \in \mathcal{S}$ for all $A_i \in X_{(c,\varphi)}$ and $t_p[A_i] = \_$ for all other attributes. Indeed, $t_p[A_i]$ cannot be a constant not in $\mathcal{S}$, since otherwise by the choice of the constants in t, t does not match the LHS of φ and thus $I \models \varphi$. However, given the form of φ and the rule of the algorithm for adding attribute-value pairs, CONSISTENCY would have added (A, a) to $\mathcal{S}$ and thus $t[A] = a$. Hence $I \models \varphi$, contradicting the assumption.

For the other direction, suppose that Σ is consistent but that the algorithm decides otherwise. Let $I = \{t\}$ such that $I \models \Sigma$. Suppose that the algorithm detects inconsistency at step i. It is easy to see that $t[A_j] = a_j$ for all $(A_j, a_j) \in \mathcal{S}_{i-1}$. Further, there must exist two CFDs $\varphi_1 = (R : X \to A, (t_p \parallel a_1))$ and $\varphi_2 = (R : Y \to A, (s_p \parallel a_2))$ in Σ such that both (A, a_1) and (A, a_2) are added to $\mathcal{S}_i$ but $a_1 \neq a_2$. Then, in order for I to satisfy Σ it must be the case that $t[A] = a_1$ and $t[A] = a_2$, which is clearly impossible. Thus the algorithm could not have detected an inconsistency and hence will return that Σ is consistent, as desired. □

Finally, we consider the case when $\mathsf{attr}(\Sigma)$ contains finite-domain attributes. Denote by $\mathsf{finattr}(R)$ the set of attributes in $\mathsf{attr}(R)$ that have a finite domain. For ease of exposition we assume *w.l.o.g.* that all attributes in $\mathsf{finattr}(R)$ have the same finite domain fdom.

PROPOSITION 3.6. *For any set Σ of CFDs defined on a schema R, determining whether Σ is consistent can be done in $O(|\mathsf{fdom}|^{|\mathsf{finattr}(R)|}|\Sigma|^2|\mathsf{attr}(\Sigma)|)$ time. If the schema R is fixed, then the complexity reduces to $O(|\Sigma|^2)$.*

PROOF. For each tuple $c \in \mathsf{fdom}^{\mathsf{finattr}(R)}$ of possible finite-domain values over the attributes in $\mathsf{finattr}(R)$, we define $\Sigma(c)$ to be the union of Σ with the set of total

CFDs $\{\varphi_A^c = (R : A \to A, (\_ \parallel c)) \mid A \in \mathsf{finattr}(R)\}$. Clearly, for any instance $I = \{t\}$ of R, $I \models \Sigma(c)$ iff $I \models \Sigma$ and $t[\mathsf{finattr}(R)] = c$. Note that the inclusion of these total CFDs allows us to assume that all attributes have an infinite domain. More precisely, we give a consistency checking algorithm as follows. For each $c \in \mathsf{fdom}^{\mathsf{finattr}(R)}$ it applies algorithm CONSISTENCY given in the proof of Proposition 3.5 to the set $\Sigma(c)$ of CFDs. If for some c, CONSISTENCY establishes the consistency of $\Sigma(c)$, then the algorithm concludes that Σ is consistent. If for all possible c, CONSISTENCY decides that $\Sigma(c)$ is inconsistent, then it concludes that Σ is inconsistent. From the correctness of CONSISTENCY it is easy to show that this algorithm is correct.

The algorithm takes at most $O(|\mathsf{fdom}|^{|\mathsf{finattr}(R)|}|\Sigma|^2|\mathsf{attr}(\Sigma)|)$ time, no larger than $O(|\mathsf{fdom}|^{|\mathsf{finattr}(R)|}|\Sigma|^2|\mathsf{attr}(R)|)$ by $|\mathsf{attr}(\Sigma)| \leq |\mathsf{attr}(R)|$. If R is fixed, $|\mathsf{fdom}|^{|\mathsf{finattr}(R)|}$ and $|\mathsf{attr}(R)|$ are constants, and the time complexity reduces to $O(|\Sigma|^2)$. □

3.2 Approximation Factor Preserving Reduction to MAXGSAT

The intractability of Theorem 3.2 highlights the need for an approximation algorithm that, given a set Σ of CFDs, finds a maximum subset of Σ that is consistent. Specifically, given Σ, the algorithm is to find a consistent subset Σ_m of Σ such that $\mathsf{card}(\Sigma_m) \geq (1-\epsilon) \cdot \mathsf{card}(\mathsf{OPT}_{maxsc}(\Sigma))$, where $\mathsf{OPT}_{maxsc}(\Sigma)$ denotes a maximum consistent subset of Σ, and ϵ is a constant referred to as the *approximation factor*. The algorithm is referred to as an ϵ-*approximation* algorithm. We refer to the problem of finding a maximal consistent subset of Σ as MAXSC.

To develop an approximation algorithm for MAXSC, we capitalize on existing approximation algorithms for a well-studied NP-complete problem, namely, the Maximum Generalized Satisfiability (MAXGSAT) problem. Given a set $\Phi = \{\phi_1, \ldots, \phi_n\}$ of Boolean expressions, the MAXGSAT problem is to find a truth assignment that satisfies the maximum number of expressions in Φ. Approximation algorithms have been developed for MAXGSAT, especially for m-MAXGSAT, namely, when each ϕ_i involves at most m Boolean variables (see, *e.g.,* [Papadimitriou 1994]). Thus it suffices to provide an *approximation factor preserving reduction* from MAXSC to MAXGSAT (see, *e.g.,* [Vazirani 2003] for approximation factor preserving reductions). To this end we provide two PTIME functions f and g such that for any set Σ of CFDs, (a) $f(\Sigma)$ is an instance Φ_Σ of MAXGSAT, and $g(\Phi_m)$ is a consistent subset of Σ if Φ_m is the set of satisfied expressions in Φ_Σ; (b) $\mathsf{card}(\mathsf{OPT}_{maxgsat}(f(\Sigma))) \geq \mathsf{card}(\mathsf{OPT}_{maxsc}(\Sigma))$; and (c) $\mathsf{card}(g(\Phi_m)) \geq \mathsf{card}(\Phi_m)$, where $\mathsf{OPT}_{maxgsat}(f(\Sigma))$ denotes the maximum set of satisfied expressions in Φ_Σ. That is, g is guaranteed to return a feasible MAXSC solution for Σ. Such a reduction ensures that if MAXGSAT has an ϵ-factor approximation algorithm, then so does MAXSC. Indeed, if $\mathsf{card}(\Phi_m) \geq (1-\epsilon) \cdot \mathsf{card}(\mathsf{OPT}_{maxgsat}(f(\Sigma)))$, then from (b) and (c) above, it follows that $\mathsf{card}(g(\Phi_m)) \geq (1-\epsilon) \cdot \mathsf{card}(\mathsf{OPT}_{maxsc}(\Sigma))$. In particular, if Φ_m is the optimal solution of MAXGSAT for $f(\Sigma)$, then $g(\Phi_m)$ is the optimal solution of MAXSC for Σ.

Reduction. To give the reduction, we first revise the notation of active domain introduced in the proof of Theorem 3.2. Assume that Σ is defined on a relation schema R, where $\mathsf{attr}(R) = \{A_1, \ldots, A_n\}$. For each $i \in [1, n]$, we define $\mathsf{adom}(A_i)$ to be the set consisting of (a) all the data values appearing in some $t_p[A_i]$ for t_p in Σ; (b) a value in $\mathsf{dom}(A_i)$ that is not yet in the set, if there exists any; note that if $\mathsf{dom}(A_i)$ is a finite domain, there may not exist such a value. Let k denote the

number of constraints in Σ. Then $\mathsf{adom}(A_i)$ has at most $k+1$ values.

The function f is defined as follows. For each $i \in [1,n]$ and $a \in \mathsf{adom}(A_i)$, we introduce a Boolean variable $x(i,a)$ such that $x(i,a) = \mathsf{true}$ if and only if $t[A_i] = a$. That is, these Boolean variables define a tuple t. In addition, for each $(R : X \to A, t_p) \in \Sigma$, we define a Boolean expression $\phi(t_p)$:

$$\phi(t_p) = (\bigvee_{B \in X} (t[B] \not\asymp t_p[B]) \vee (t[A] \asymp t_p[A])) \wedge \Psi_R,$$

where $t[B] \asymp t_p[B]$ can be written as $x(i,a)$ if $B = A_i$ and $t_p[B] = a$, and as true if $t_p[B] =$ '_'; similarly, $t[B] \not\asymp t_p[B]$ can be written as $\overline{x(i,a)}$ if $B = A_i$ and $t_p[B] = a$, and as false if $t_p[B] =$ '_'. The expression Ψ_R is defined as follows, which is to assure that for each $i \in [1,n]$, $t[A_i]$ has a unique value:

$$\Psi_R = \bigwedge_{i\in[1,n]} \bigwedge_{a\in\mathsf{adom}(A_i)} ((x(i,a) \to \bigwedge_{b\in\mathsf{adom}(A_i) \wedge b\neq a} \overline{x(i,b)}) \wedge (\overline{x(i,a)} \to \bigvee_{b\in\mathsf{adom}(A_i) \wedge b\neq a} x(i,b))).$$

We define $f(\Sigma) = \Phi_\Sigma = \{\phi(t_p) \mid t_p \in \Sigma\}$. Note that each expression in Φ_Σ has at most $n \cdot (k+1)$ Boolean variables.

We define the function g as follows. For a truth assignment ρ for Φ_Σ, let Φ_m be the set of satisfied expressions in Φ_Σ by ρ. We instantiate t based on ρ as follows: $t[A_i] = a$ if and only if $\rho(x(i,a)) = \mathsf{true}$. Then $g(\Phi_m)$ is defined to be the set of CFDs satisfied by t. It is easy to verify that $\mathsf{card}(\Phi_m) = \mathsf{card}(g(\Phi_m))$.

Verification. We next show that the reduction is approximation factor preserving. First observe that functions f and g can be computed in PTIME in $|\Sigma|$ and n. Second, from the discussion above it follows that $\mathsf{card}(\mathsf{OPT}_{maxgsat}(f(\Sigma))) = \mathsf{card}(\mathsf{OPT}_{maxsc}(\Sigma))$. Third, for any truth assignment ρ for Φ_Σ, if Φ_m is the set of satisfied expressions in Φ_Σ by ρ, then $\mathsf{card}(\Phi_m) = \mathsf{card}(g(\Phi_m))$, for the same reason given above. Taken together, the reduction is indeed approximation factor preserving. This allows us to derive approximation algorithms for MAXSC from existing approximation algorithms for MAXGSAT.

4. IMPLICATION ANALYSIS OF CONDITIONAL FUNCTIONAL DEPENDENCIES

In this section, we study the implication problem for CFDs. The *implication problem for* CFDs is to determine, given a set Σ of CFDs and a single CFD φ on a relation schema R, whether or not Σ entails φ, denoted by $\Sigma \models \varphi$, *i.e.,* whether or not for all instances I of R, if $I \models \Sigma$ then $I \models \varphi$. We consider consistent Σ only.

We show that the richer semantics of CFDs complicates the implication analysis: as opposed to standard FDs, the implication problem for CFDs is coNP-complete in general. We provide a sound and complete inference system for the implication analysis of CFDs, as a nontrivial extension of Armstrong's Axioms for FDs. Based on these we present an algorithm for computing a minimal cover of a set of CFDs.

4.1 An Inference System for Conditional Functional Dependencies

Armstrong's Axioms for FDs are found in almost every database textbook, and are fundamental to the implication analysis of FDs. Analogous to Armstrong's Axioms, we provide an inference system for CFDs, denoted by $\mathcal{I}$, in Fig. 3. Given a finite set $\Sigma \cup \{\varphi\}$ of CFDs, we use $\Sigma \vdash_{\mathcal{I}} \varphi$ to denote that φ is provable from Σ using $\mathcal{I}$.

FD1:	If $A \in X$, then $(R : X \to A,\ t_p)$, where $t_p[A_L] = t_p[A_R]$ = 'a' for some 'a' $\in \mathsf{dom}(A)$, or both are equal to a '_'.
FD2:	If (1) $(R : X \to A_i, t_i)$ such that $t_i[X] = t_j[X]$ for all $i, j \in [1, k]$, (2) $(R : [A_1, \ldots, A_k] \to B, t_p)$ and moreover, (3) $(t_1[A_1], \ldots, t_k[A_k]) \preceq t_p[A_1, \ldots, A_k]$, then $(R : X \to B, t'_p)$, where $t'_p[X] = t_1[X]$ and $t'_p[B] = t_p[B]$.
FD3:	If $(R : [B, X] \to A, t_p)$, $t_p[B]$ ='_', and $t_p[A]$ is a constant, then $(R : X \to A, t'_p)$, where $t'_p[X \cup \{A\}] = t_p[X \cup \{A\}]$.
FD4:	If (1) $\Sigma \vdash_{\mathcal{I}} (R : [X, B] \to A, t_i)$ for $i \in [1, k]$, (2) $\mathsf{dom}(B) = \{b_1, \ldots, b_k, b_{k+1}, \ldots, b_m\}$, and $(\Sigma, B = b_l)$ is not consistent except for $l \in [1, k]$, and (3) for $i, j \in [1, k]$, $t_i[X] = t_j[X]$, and $t_i[B] = b_i$, then $\Sigma \vdash_{\mathcal{I}} (R : [X, B] \to A, t_p)$ where $t_p[B]$ ='_' and $t_p[X] = t_1[X]$.

Fig. 3. Inference Rules for CFDs

Example 4.1. Consider the set Σ of CFDs consisting of $\psi_1 = (R : [A, B] \to C, (a, b_1 \parallel c))$, $\psi_2 = (R : [A, B] \to C, (a, b_2 \parallel c))$, $\psi_3 = (R : [C, D] \to E, (\_, \_ \parallel \_))$ and $\psi_4 = (R : B \to B, (b_3 \parallel b_2))$, where $\mathsf{dom}(B) = \{b_1, b_2, b_3\}$. Let $\varphi = (R : [A, D] \to E, (a, \_ \parallel \_))$. Then $\Sigma \vdash_{\mathcal{I}} \varphi$ can be proved as follows (the notion that $(\Sigma, B = b_3)$ is not consistent will be elaborated shortly):

(1) $(R : [A, B] \to C, (a, b_1 \parallel c))$	ψ_1
(2) $(R : [A, B] \to C, (a, b_2 \parallel c))$	ψ_2
(3) $(R : [A, B] \to C, (a, \_ \parallel c))$	(1), (2) and FD4; $((\Sigma, B = b_3)$ is not consistent)
(4) $(R : A \to C, (a \parallel c))$	(3) and FD3
(5) $(R : [A, D] \to A,\ (a, \_ \parallel a))$	FD1
(6) $(R : [A, D] \to D,\ (a, \_ \parallel \_))$	FD1
(7) $(R : [A, D] \to C,\ (a, \_ \parallel c))$	(4), (5) and FD2
(8) $(R : [C, D] \to E,\ (\_, \_ \parallel \_))$	ψ_3
(9) $(R : [A, D] \to E,\ (a, \_ \parallel \_))$	(6), (7), (8) and FD2

While the rules FD1 and FD2 in $\mathcal{I}$ are extensions of Armstrong's Axioms for FDs, FD3 and FD4 do not find a counterpart in Armstrong's Axioms. We next illustrate the inference rules in $\mathcal{I}$ and show their soundness. That is, if $\Sigma \vdash_{\mathcal{I}} \varphi$ then $\Sigma \models \varphi$.

FD1 and FD2 extend Armstrong's Axioms of reflexivity and transitivity, respectively. The rule FD1 is self-explanatory and is illustrated on lines 5 and 6 in Example 4.1.

In contrast, in order for FD2 to cope with pattern tuples which are not found in FDs, it employs an order relation $\preceq$, defined as follows. For a pair η_1, η_2 of constants or '_', we say that $\eta_1 \preceq \eta_2$ if either $\eta_1 = \eta_2 = a$ where a is a constant, or $\eta_2 =$ '_'. The $\preceq$ relation naturally extends to pattern tuples. For instance, $(a, b) \preceq (\_, b)$. Intuitively, the use of $\preceq$ in FD2 assures that $(t_1[A_1], \ldots, t_k[A_k])$ is in the "scope" of $t_p[A_1, \ldots, A_k]$, *i.e.,* the pattern $t_p[A_1, \ldots, A_k]$ is applicable. In Example 4.1, FD2 can be applied on line 9 because the pattern tuple $t_1[C, D] = (c, \_)$ obtained from lines 6 and 7, and the pattern tuple $t_2[C, D] = (\_, \_)$ on the LHS of ψ_3 (line 8), satisfy $t_1[C, D] \preceq t_2[C, D]$. To see that FD2 is sound, note that any tuple t that matches a pattern tuple t_p also matches any pattern tuple t'_p if $t_p \preceq t'_p$. More specifically, assume that conditions (1), (2) and (3) of the rule hold. Let Σ consist of $\varphi_i = (R : X \to A_i, t_i)$, $\phi = (R : [A_1, \ldots, A_k] \to B, t_p)$ and $\psi = (R : X \to B, t'_p)$, for $i \in [1, k]$. Assume that $\Sigma \vdash_{\mathcal{I}} \psi$ by FD2. We need to show that for any instance I of R, if $I \models \Sigma$, then $I \models \psi$. Indeed, for any two tuples $s, t \in I$, if $s[X] = t[X] \asymp t'_p[X]$, then from condition (1) and the assumption that $t'_p[X] = t_1[X]$ it follows that

$s[X] = t[X] \asymp t_i[X]$ and $s[A_i] = t[A_i] \asymp t_i[A_i]$ for $i \in [1, k]$. By condition (3), we have that $s[A_1, \ldots, A_k] = t[A_1, \ldots, A_k] \asymp t_p[A_1, \ldots, A_k]$. Thus from condition (2) (*i.e.,* $I \models \phi$) it follows that $s[B] = t[B] \asymp t_p[B] = t'_p[B]$. Hence $I \models \psi$, as desired.

FD3 tells us that for a CFD $\varphi = (R : [B, X] \to A, t_p)$, if $t_p[B] =$ '_' and $t_p[A]$ is a constant 'a', then φ can be simplified by dropping the B attribute from the LHS of the embedded FD. To see this, consider an instance I of R such that $I \models \varphi$, and *any* tuple t in I. Note that since $t_p[B] =$ '_', if $t[X] \asymp t_p[X]$ then $t[B, X] \asymp t_p[B, X]$ and $t[A]$ has to be 'a' regardless of what value $t[B]$ has. Thus φ entails $(R : X \to A, t_p)$, and $I \models (R : X \to A, t_p)$. This rule is illustrated on line 4 in Example 4.1.

FD4 deals with attributes of finite domains, which are not an issue for standard FDs since FDs have no pattern tuples. They are given *w.r.t.* a set Σ of CFDs. More specifically, to use this rule one needs to determine, given Σ on a relation schema R, an attribute B in $\mathsf{attr}(R)$ with a finite domain and a constant $b \in \mathsf{dom}(B)$, whether or not there exists an instance I of R such that $I \models \Sigma$ and moreover, there is a tuple t in I such that $t[B] = b$. We say that $(\Sigma, B = b)$ is *consistent* if and only if such an instance I exists. That is, since the values of B have finitely many choices, we need to find out for which $b \in \mathsf{dom}(B)$, Σ and $B = b$ make sense when put together. For example, consider the set $\Sigma = \{\psi_2, \psi_3\}$ given in Example 3.1, and the bool attribute A. Then neither $(\Sigma, A = \mathsf{true})$ nor $(\Sigma, A = \mathsf{false})$ is consistent. FD4 says that for an attribute B of a finite domain and *w.r.t.* a given set Σ of CFDs, if $\Sigma \vdash_{\mathcal{I}} (R : [X, B] \to A, t_i)$ when $t_i[B]$ ranges over all $b \in \mathsf{dom}(B)$ such that $(\Sigma, B = b)$ is consistent, then $t_i[B]$ can be "upgraded" to '_'. That is, for any instance I, if $I \models \Sigma$, then $I \models (R : [X, B] \to A, t_p)$, where $t_p[B] =$ '_'. Indeed, suppose that $I \models \Sigma$ but $I \not\models (R : [X, B] \to A, t_p)$. Suppose that $t_p[A] =$ _ (similarly for the case $t_p[A] = a \in \mathsf{dom}(A)$). Then there exist $s, t \in I$ such that $\{s, t\} \models \Sigma$, $s[X, B] = t[X, B] \asymp t_p[X, B]$ but $s[A] \neq t[A]$. Let $b = s[B]$ (and hence $b = t[B]$). Then $\{s, t\} \not\models (R : [X, B] \to A, t_i)$ where $t_i[B] = b$. This contradicts the assumption that $\Sigma \vdash_{\mathcal{I}} (R : [X, B] \to A, t_i)$ for $t_i[B]$ ranging over all such $b \in \mathsf{dom}(B)$ that $(\Sigma, B = b)$ is consistent. Thus $I \models (R : [X, B] \to A, t_p)$, where $t_p[B] =$ '_'.

One might wonder why $\mathcal{I}$ has no rule analogous to the augmentation rule in Armstrong's Axioms. The reason is that we assume that all our CFDs are in the normal form and therefore only have a single attribute in their RHS. As a result, there is no need for augmenting the RHS with attributes. However, if one would lift this assumption then (four) additional rules need to be added to $\mathcal{I}$ that allow to transform any CFD into its normal form and back. These rules are straightforward (they basically simulate the construction from Σ to Σ_{nf} and back as indicated in the proof of Proposition 2.3) and do not provide any more insight in the intricacies involved in the implication problem for CFDs. Since it suffices to show the completeness of $\mathcal{I}$ for CFDs in the normal form, we omit a detailed description of these additional rules and assume, as usual, that all CFDs in Σ are in the normal form.

From the discussion above it follows that if $\Sigma \vdash_{\mathcal{I}} \varphi$ then $\Sigma \models \varphi$, or in other words the set of inference rules $\mathcal{I}$ is *sound*. The theorem below tells us that, analogous to Armstrong's Axioms for FDs, the set of inference rules of $\mathcal{I}$ is also *complete*, *i.e.,* if $\Sigma \models \varphi$ then $\Sigma \vdash_{\mathcal{I}} \varphi$. Hence, $\mathcal{I}$ characterizes the implication analysis for CFDs.

THEOREM 4.2. *The inference system $\mathcal{I}$ is sound and complete for implication of*

CFD*s*.

PROOF. The soundness of the rules follows from the discussion given above. We focus on their completeness. For a set Σ of CFDs, let Σ_c be the set of CFDs in Σ of the form $(R : X \to A, (t_p \parallel a))$ for some constant $a \in \mathsf{dom}(A)$, and Σ_v be the set of CFDs in Σ of the form $(R : X \to A, (t_p \parallel \_))$. We denote by $\mathsf{finattr}(R)$ the set of attributes in $\mathsf{attr}(R)$ that have a finite domain fdom, and by $\mathsf{attr}(\Sigma)$ the set of attributes in $\mathsf{attr}(R)$ that appear in some CFDs in Σ.

We show the completeness of $\mathcal{I}$ in the following steps: **(1.a)** first, we show that if $\Sigma \models \varphi$ then $\Sigma \vdash_{\mathcal{I}} \varphi$ when $\varphi = (R : X \to A, (t_p \parallel a))$ for some constant $a \in \mathsf{dom}(A)$ and when $\mathsf{attr}(\Sigma)$ does not contain any finite domain attributes; **(1.b)** next, while we still assume that φ has a constant RHS, we allow finite domain attributes in $\mathsf{attr}(\Sigma)$; **(2.a)** we then show that $\Sigma \models \varphi$ implies $\Sigma \vdash_{\mathcal{I}} \varphi$ when $\varphi = (R : X \to A, (t_p \parallel \_))$ and when $\mathsf{attr}(\Sigma)$ does not contain any finite domain attributes; and finally **(2.b)** we still assume that φ has a variable RHS but allow finite domain attributes in $\mathsf{attr}(\Sigma)$. All four steps are proven by extending the standard proof of the completeness of Armstrong's Axioms for FDs (see, *e.g.,* [Abiteboul et al. 1995]). That is, we first provide an algorithm that computes a so-called *closure set* which characterizes when a set of CFDs implies a given CFD; and then we show that the computation of the closure set can be simulated using rules in $\mathcal{I}$.

(1.a) Let $\varphi = (R : X \to A, (t_p \parallel a))$ for some $a \in \mathsf{dom}(A)$ and assume that $\mathsf{attr}(\Sigma)$ does not have any finite domain attributes. We first observe that $\Sigma \models \varphi$ iff $\Sigma_c \models \varphi$. Clearly, since $\Sigma_c \subseteq \Sigma$, $\Sigma_c \models \varphi$ implies $\Sigma \models \varphi$. Conversely, assume that $\Sigma \models \varphi$ but that $\Sigma_c \not\models \varphi$. Then there must exist a single-tuple instance $I = \{t\}$ such that $I \models \Sigma_c$ but $I \not\models \varphi$. However, since single-tuple instances trivially satisfy any CFD in Σ_v, $I \models \Sigma_v$ as well. Therefore, $I \models \Sigma$ but $I \not\models \varphi$. This contradicts the assumption that $\Sigma \models \varphi$. Hence, $\Sigma_c \models \varphi$ implies $\Sigma \models \varphi$. As a result, it suffices to only consider Σ_c instead of Σ for the implication problem for CFDs φ with a constant RHS.

Next, we provide an algorithm, called c-CLOSURE, that takes as input Σ_c, X and t_p, and outputs a set of attribute-value pairs, denoted by $\Sigma_c^*(X, t_p)$. Analogous to the closure set of standard FDs, the set $\Sigma_c^*(X, t_p)$ satisfies the following property: $(A, a) \in \Sigma_c^*(X, t_p)$ iff $\Sigma \models (R : X \to A, (t_p \parallel a))$. As shown in Fig. 4, c-CLOSURE is similar to the algorithm for computing the closure of FDs except that (i) it takes into account the pattern tuple t_p over X; (ii) it returns a set of attribute-value pairs instead of attributes only; (iii) *result* is initialized using the attributes in X for which t_p is a constant; and (iv) it uses a different "transitivity" rule (lines 4-5).

Before we show that $\Sigma \models \varphi$ implies $\Sigma \vdash_{\mathcal{I}} \varphi$ we establish the key property of $\Sigma_c^*(X, t_p)$. That is, $(A, a) \in \Sigma_c^*(X, t_p)$ iff $\Sigma \models (R : X \to A, (t_p \parallel a))$.

First, suppose that $\Sigma \models \varphi$ but $(A, a) \notin \Sigma_c^*(X, t_p)$. Consider the single-tuple instance $I = \{t\}$ such that $t[A_i] = a_i$ for all $(A_i, a_i) \in \Sigma_c^*(X, t_p)$ and all other attributes in t are instantiated with constants not appearing in the attributes of any pattern tuple in $\Sigma \cup \{\varphi\}$. Note that this is possible since Σ is consistent and all attributes in $\mathsf{attr}(\Sigma)$ are assumed to have an infinite domain. We now show that $I \models \Sigma$ but $I \not\models \varphi$. We clearly have that $I \models \Sigma_v$ since I is a single-tuple instance. Moreover, for any $\psi = (R : Y \to B, (s_p \parallel b)) \in \Sigma_c$ such that for all $B_i \in Y$ either $(B_i, s_p[B_i]) \in \Sigma_c^*(X, t_p)$ or $s_p[B_i] = \_$, we have that $(B, b) \in \Sigma_c^*(X, t_p)$, and thus $I \models \psi$ for any such ψ. For CFDs ψ' that are not of the above form, it suffices to

Input: A set Σ_c of CFDs, a set of attributes X and pattern t_p.
Output: The closure set $\Sigma_c^*(X, t_p)$.

1. *unused* := Σ_c;
2. *result* := $\{(A_i, t_p[A_i]) \mid A_i \in X, t_p[A_i] \in \mathsf{dom}(A_i)\}$;
3. **repeat until** no further change
4. **if** $(R : Y \to B, (s_p \parallel b)) \in$ *unused* **and** (for each $B_i \in Y$,
5. either $(B_i, s_p[B_i]) \in$ *result* or $s_p[B_i] = \_$) **then do**
6. (i) *unused* := *unused* $\setminus \{(R : Y \to B, (s_p \parallel b))\}$;
7. (ii) *result* := *result* $\cup \{(B, b)\}$;
8. **output** *result*

Fig. 4. Algorithm c-CLOSURE

observe that I does not match LHS(ψ') and therefore trivially satisfies ψ'. Hence, we may conclude that $I \models \Sigma$. However, since $t[A] \neq a$ we have indeed that $I \not\models \varphi$ which contradicts the assumption that $\Sigma \models \varphi$.

Conversely, assume that $(A, a) \in \Sigma_c^*(X, t_p)$ but $\Sigma \not\models \varphi$. Then there must exist a single-tuple instance $I = \{t\}$ such that $I \models \Sigma$, $t[X] \asymp t_p$ but $t[A] \neq a$. However, from $I \models \Sigma$ and $t[X] \asymp t_p$ it follows that $t[A_i] = a_i$ for all $(A_i, a_i) \in \Sigma_c^*(X, t_p)$. In particular, $t[A]$ must be 'a', contradicting the assumption that $I \not\models \varphi$. Thus $\Sigma \models \varphi$.

We are now ready to show that if $\Sigma \models \varphi$ then $\Sigma \vdash_{\mathcal{I}} \varphi$. Again, the proof is similar to its standard FD counterpart. That is, we show that $\Sigma \vdash_{\mathcal{I}} (R : X \to A_i, t_p \parallel a_i))$ for all $(A_i, a_i) \in \Sigma_c^*(X, t_p)$. Intuitively, we "verify" algorithm c-CLOSURE using the inference rules in $\mathcal{I}$. Denote by *result*$_i$ the content of *result* after i iterations of c-CLOSURE on input Σ, X and t_p. More specifically, we denote by $A^{(i)}$ the set of attributes in *result*$_i$ and by $a^{(i)}$ the corresponding tuple of constants associated with these attributes. The set *result*$_0$ is initialized by line 2 in Figure 4.

We show by induction on i that for each $(A_j, a_j) \in$ *result*$_i$, $\Sigma \vdash_{\mathcal{I}} (R : X \to A_j, t_p \parallel a_j))$. For each $(A_j, a_j) \in$ *result*$_0$, we have that $\Sigma \vdash_{\mathcal{I}} (R : X \to A_j, (t_p \parallel a_j))$ by FD1. Suppose inductively that we have proofs $\sigma_1^j, \ldots, \sigma_{k_i}^j$ of $(R : X \to A_j, (t_p \parallel a_j))$ for all $(A_j, a_j) \in$ *result*$_i$. Suppose that $(R : Y \to B, (s_p \parallel b)) \in \Sigma$ is chosen for the $(i{+}1)$th iteration and that it passes the condition stated on lines 4-5. In other words, either $(B_i, s_p[B_i]) \in$ *result*$_i$ or $s_p[B_i] = \_$. As a result, *result*$_{i+1}$ = *result*$_i \cup \{(B, b)\}$. We therefore extend the proof by adding the following steps:

(1) $(R : Y \to B, (s_p \parallel b))$	in Σ;
(2) $(R : Y' \to B, (s_p[Y'] \parallel b))$	by FD3; $s_p[Y']$ consists of constants only;
(3) $(R : A^{(i)} \to B_j, (a^{(i)} \parallel s_p[B_j]))$	by FD1; for each $B_j \in Y'$;
(4) $(R : A^{(i)} \to B, (a^{(i)} \parallel b))$	by (2), (3) and FD2;
(5) $(R : X \to A_j, (t_p \parallel a^{(i)}[A_j]))$	induction hypothesis; for each $A_j \in A^{(i)}$;
(6) $(R : X \to B, (t_p \parallel b))$	by (4), (5) and FD2.

From this, we may conclude that we have a proof for $(R : X \to A_j, (t_p \parallel a_j))$ for all $(A_j, a_j) \in$ *result*$_{i+1}$. Proceed in this way until the completion of *result*, we indeed have a proof for $(R : X \to A_i, (t_p \parallel a_i))$ for all $(A_i, a_i) \in \Sigma_c^*(X, t_p)$. In particular, since $\Sigma \models \varphi$ and therefore $(A, a) \in \Sigma_c^*(X, t_p)$, we have a proof for $(R : X \to A, (t_p \parallel a))$, as desired. We remark that only rules FD1, FD2 and FD3 are needed in this case. Moreover, it is easy to verify that c-CLOSURE provides an

algorithm for deciding $\Sigma \models \varphi$ that runs in $O(|\Sigma|^2|\mathsf{attr}(\Sigma)|)$ time.

(1.b) Let $\varphi = (R : X \to A, (t_p \parallel a))$ for some $a \in \mathsf{dom}(A)$ and assume that $\mathsf{attr}(\Sigma)$ contains finite domain attributes. We first show how c-CLOSURE can be used to decide $\Sigma \models \varphi$ in this case. For a given set X of attributes, let X' be $X \cup \mathsf{finattr}(R)$. For a pattern tuple t_p over X, let c_p be a tuple of constants over $\mathsf{finattr}(R)$, *i.e.,* $c_p \in \mathsf{fdom}^{\mathsf{finattr}(R)}$, such that (i) c_p and t_p have the same constants on the attributes in $\mathsf{finattr}(R)$, and (ii) c_p and Σ are consistent. Denote by $c_p \bowtie t_p$ the pattern tuple over X' that is the same as t_p on X and has the same constants as c_p on $\mathsf{finattr}(R)$.

We can see that $(A, a) \in \bigcap_{c_p} \Sigma_c^*(X', c_p \bowtie t_p)$ iff $\Sigma \models \varphi$. Here c_p ranges over all constant tuples as described above and $\Sigma_c^*(X', c_p \bowtie t_p)$ is the result of c-CLOSURE on input Σ, X' and $c_p \bowtie t_p$. Intuitively, the enumeration of constant tuples of attributes in $\mathsf{finattr}(R)$ allows us to reduce case **(1.b)** to **(1.a)**. The intersection filters out any attribute-value pair inferred due to the presence of specific constants in $\mathsf{finattr}(R)$. Clearly, the "if"-direction follows from the fact that if $\Sigma \models (R : X \to A, (t_p \parallel a))$ then $\Sigma \models (R : X' \to A, (c_p \bowtie t_p \parallel a))$ for all c_p. For the "only if"-direction, suppose that (A, a) is in the intersection but $\Sigma \not\models \varphi$. Then there must exist a single-tuple instance $I = \{t\}$ such that $I \models \Sigma$ but $I \not\models \varphi$. Let $c_p = t[\mathsf{finattr}(R)]$ (note that c_p and Σ are consistent). Then, I serves as a counterexample for $\Sigma \models (R : X' \to A, (c_p \bowtie t_p \parallel a))$ and thus $(A, a) \notin \Sigma_c^*(X', c_p \bowtie t_p)$. Hence, (A, a) cannot be in $\bigcap_{c_p} \Sigma_c^*(X', c_p \bowtie t_p)$ either, contradicting our assumption.

We use the following algorithm to compute $\bigcap_{c_p} \Sigma_c^*(X', c_p \bowtie t_p)$: (i) for each c_p, execute c-CLOSURE on input Σ, X' and $c_p \bowtie t_p$; and (ii) take the intersection of all result sets computed in (i). Given the complexity of c-CLOSURE, this algorithm runs in $O(|\mathsf{fdom}|^{|\mathsf{finattr}(R)|}|\Sigma|^2|\mathsf{attr}(\Sigma)|)$ time.

For the completeness of $\mathcal{I}$, we construct proofs for $(R : X \to A_i, (t_p \parallel a_i))$ for all $(A_i, a_i) \in \bigcap_{c_p} \Sigma_c^*(X', c_p \bowtie t_p)$. That is, for each $(A_i, a_i) \in \bigcap_{c_p} \Sigma_c^*(X', c_p \bowtie t_p)$, we extend the inductive $\mathcal{I}$-proofs given in case **(1.a)** as follows:

(1) $(R : X' \to A_i, (c_p \bowtie t_p \parallel a_i))$	for each c_p consistent with Σ; and by inductive proofs as in case **(1.a)**;
(2) $(R : X' \to A_i, (\_, \ldots, \_, t_p \parallel a_i))$	by (1) and FD4;
(3) $(R : X \to A_i, (t_p \parallel a_i))$	by (2) and FD3.

In particular, this shows that $\Sigma \vdash_{\mathcal{I}} (R : X \to A, (t_p \parallel a))$ since $\Sigma \models (R : X \to A, (t_p \parallel a))$ implies $(A, a) \in \bigcap_{c_p} \Sigma_c^*(X', c_p \bowtie t_p)$.

(2.a) Let $\varphi = (R : X \to A, (t_p \parallel \_))$ and assume that $\mathsf{attr}(\Sigma)$ does not have finite domain attributes. In this case we separate the treatment of CFDs in Σ_c and the treatment of Σ_v, as follows. First, we compute $\Sigma_c^*(X, t_p)$ as described in case (**1.a**). Note that the attribute-value pairs induced in this phase are bindings necessarily enforced by Σ_c. Second, leveraging $\Sigma_c^*(X, t_p)$, we simplify the treatment of Σ_v. More specifically, we expand X and t_p to $\overline{X}$ and $\overline{t_p}$ by incorporating attributes and constants induced from $\Sigma_c^*(X, t_p)$, as follows. Initially, let $\overline{X}$ be X and $\overline{t_p}$ be t_p. For each $(A_i, a_i) \in \Sigma_c^*(X, t_p)$, we add A_i to $\overline{X}$ and extend $\overline{t_p}$ by letting $\overline{t_p}[A_i] = a_i$. As will be shown shortly, $\Sigma \models (R : X \to A, (t_p \parallel \_))$ iff $\Sigma_v \models (R : \overline{X} \to A, (\overline{t_p} \parallel \_))$. Therefore, *w.l.o.g.* we may assume $\varphi = (R : \overline{X} \to A, (\overline{t_p} \parallel \_))$ and consider Σ_v only.

We now show that $\Sigma \models (R : X \to A, (t_p \parallel \_))$ iff $\Sigma_v \models (R : \overline{X} \to A, (\overline{t_p} \parallel \_))$.

First assume that $\Sigma \models (R : X \rightarrow A, (t_p \parallel \_))$ but there exists an instance I of R such that $I \models \Sigma_v$ but there exist two tuples $s, t \in I$ such that $s[\overline{X}] = t[\overline{X}] \asymp \overline{t_p}[\overline{X}]$, while $s[A] \neq t[A]$. We then construct an instance $I' = \{s', t'\}$ such that $I' \models \Sigma$ but $I' \not\models (R : X \rightarrow A, (t_p \parallel \_))$, and show that it leads to a contradiction to the assumption. To do this, we transform s and t into s' and t', respectively, as follows. First, we let $s'[\overline{X}] = s[\overline{X}]$ and $t'[\overline{X}] = t[\overline{X}]$. All other attributes in s' and t' are instantiated with constants not appearing in the attributes of any pattern tuple in $\Sigma \cup \varphi$. Moreover, for any attribute $B \in \mathsf{attr}(R)$ we guarantee that $s'[B] = t'[B]$ iff $s[B] = t[B]$. Clearly, $I' \not\models (R : X \rightarrow A, (t_p \parallel \_))$. We now show that $I' \models \Sigma$, contradicting the assumption that $\Sigma \models (R : X \rightarrow A, (t_p \parallel \_))$. First, consider $\psi = (R : Y \rightarrow B, (s_p \parallel b)) \in \Sigma_c$. From the definition of $\overline{X}$ and $\overline{t_p}$ we know that if for all $B_i \in Y$ either $(B_i, s_p[B_i]) \in \Sigma_c^*(X, t_p)$ or $s_p[B_i] = \_$, then $B \in \overline{X}$ and $\overline{t_p}[B] = b$. Therefore, $I' \models \psi$ for any such ψ in Σ_c. Moreover, for any other CFD ψ in Σ_c it follows from the choice of constants in s' and t' that I' does not satisfy $\textsc{lhs}(\psi)$ and ψ is therefore trivially satisfied by I'. Hence, $I' \models \Sigma_c$. It remains to show that $I' \models \Sigma_v$. However, this follows immediately from (i) the choice of constants in s' and t'; (ii) the fact that $s'[B] = t'[B]$ iff $s[B] = t[B]$ for all $B \in \mathsf{attr}(R)$; and (iii) $I \models \Sigma_v$. Hence, we have that $I' \models \Sigma_v$ and therefore $I' \models \Sigma$, as desired.

Conversely, assume that $\Sigma_v \models (R : \overline{X} \rightarrow A, (\overline{t_p} \parallel \_))$ but that there exists an instance $I = \{s, t\}$ such that $I \models \Sigma$, $s[X] = t[X] \asymp t_p[X]$, while $s[A] \neq t[A]$. However, this implies that $I \models (R : X \rightarrow A_i, (t_p \parallel a_i))$ for all $(A_i, a_i) \in \Sigma_c^*(X, t_p)$. In other words, $s[\overline{X}] = t[\overline{X}] \asymp \overline{t_p}$ and hence I serves as a counterexample for $\Sigma_v \models (R : \overline{X} \rightarrow A, (\overline{t_p} \parallel \_))$. We may therefore conclude that $\Sigma \models (R : X \rightarrow A, (t_p \parallel \_))$.

Given this, we present an algorithm, referred to as v-Closure, to compute a set $\Sigma_v^*(\overline{X}, \overline{t_p})$ of attributes, such that $A \in \Sigma_v^*(\overline{X}, \overline{t_p})$ iff $\Sigma \models (R : \overline{X} \rightarrow A, (\overline{t_p} \parallel \_))$. Algorithm v-Closure is the same as c-Closure, except the following: (i) it returns a set of attributes instead of attribute-value pairs; (ii) it takes Σ_v, $\overline{X}$ and $\overline{t_p}$ as input; (iii) *result* is initialized with *all* attributes in $\overline{X}$; and (iv) the "transitivity" rule requires *all* attributes $B_i \in Y$ to be in *result* and that $\overline{t_p} \preceq s_p[Y]$. It is easy to verify, in exactly the same way as in the standard FD counterpart, that $A \in \Sigma_v^*(\overline{X}, \overline{t_p})$ iff $\Sigma \models (R : \overline{X} \rightarrow A, (\overline{t_p} \parallel \_))$. Moreover, the induction proof for the completeness of $\mathcal{I}$ given for case (**1.a**) can be extended in exactly the same way as in the standard FD case. In fact, only FD1 and FD2 are needed. We remark that v-Closure provides a decision algorithm for $\Sigma \models \varphi$ that runs in $O(|\Sigma|^2 |\mathsf{attr}(\Sigma)|)$ time.

(2.b) Finally, let $\varphi = (R : X \rightarrow A, (t_p \parallel \_))$ and assume that $\mathsf{attr}(\Sigma)$ has finite domain attributes. As in case **(2.a)** we expand X and t_p to $\overline{X}$ and $\overline{t_p}$, respectively, but by using the attribute-value pairs in $\bigcap_{c_p} \Sigma_c^*(X', c_p \bowtie t_p)$. As in the proof above, one can verify that $\Sigma \models (R : X \rightarrow A, (t_p \parallel \_))$ iff $\Sigma_v \models (R : \overline{X} \rightarrow A, (\overline{t_p} \parallel \_))$.

Along the same lines as the proof of case **(1.b)**, we reduce case **(2.b)** to **(2.a)**. That is, we provide an algorithm for deciding $\Sigma_v \models (R : \overline{X} \rightarrow A, (\overline{t_p} \parallel \_))$ using the algorithm v-Closure described in case **(2.a)**. However, in contrast to the proof of case **(1.b)**, just taking the intersection of $\Sigma_v^*(\overline{X}', c_p \bowtie \overline{t_p})$ over all c_p no longer works here. To cope with this, we inductively define the following sets of attributes. First, we introduce some additional notation. For a set Y of attributes in $\mathsf{attr}(R)$,

we denote by Y_f the set of attributes $Y \cap \mathsf{finattr}(R)$. We then define $\overline{X}^{(0)} = \overline{X}$, and $\overline{X}^{(i)} = \bigcap_{c_p} \Sigma_v^*(\overline{X} \cup \overline{X}_f^{(i-1)}, c_p[\overline{X}_f^{(i-1)}] \bowtie \overline{t_p})$ for $i > 0$. As before c_p ranges over all tuples of constants over $\mathsf{finattr}(R)$ that are consistent with Σ and match $\overline{t_p}$.

We now establish the following two key properties. First, one can verify that there exists $n \leq |\mathsf{finattr}(R)|$ such that $\overline{X}^{(n+1)} = \overline{X}^{(n)}$. Second, $A \in \overline{X}^{(n)}$ iff $\Sigma \models (R : \overline{X} \rightarrow A, (\overline{t_p} \parallel \_))$.

The first property is shown as follows. First observe that $\overline{X}^{(i)} \subseteq \overline{X}^{(i+1)}$ for each $i \geq 0$. Moreover, $\overline{X}^{(i+1)} = \overline{X}^{(i)}$ if $\overline{X}_f^{(i+1)} = \overline{X}_f^{(i)}$. Assume that $X^{(i+1)} \neq X^{(i)}$. Then, in the worst case, $X^{(i+1)} \setminus X^{(i)}$ contains at most a single finite-domain attribute. Since there are only $|\mathsf{finattr}(R)|$ finite-domain attributes, it takes at most $|\mathsf{finattr}(R)|$ steps in order to get $\overline{X}_f^{(i+1)} = \overline{X}_f^{(i)}$. Hence, we know for sure that $\overline{X}^{(n+1)} = \overline{X}^{(n)}$ for some $n \leq |\mathsf{finattr}(R)|$

The second property, *i.e.,* $A \in \overline{X}^{(n)}$ iff $\Sigma \models (R : \overline{X} \rightarrow A, (\overline{t_p} \parallel \_))$ is shown as follows. Since $\Sigma \models (R : \overline{X} \rightarrow A, (\overline{t_p} \parallel \_))$ implies $\Sigma \models (R : [\overline{X}, \overline{X}_f^{(n)}] \rightarrow A, (c_p[\overline{X}_f^{(n)}] \bowtie \overline{t_p} \parallel \_))$ for each c_p, the "if"-direction is immediate. For the "only if"-direction, we show by induction on i that if $A \in \overline{X}^{(i)}$ then $\Sigma \models (R : \overline{X} \rightarrow A, (\overline{t_p} \parallel \_))$. For $i = 0$, this trivially holds. Let $i > 0$ and let $A \in \overline{X}^{(i)} \setminus \overline{X}^{(i-1)}$. Assume by contradiction that $\Sigma \not\models (R : \overline{X} \rightarrow A, (\overline{t_p} \parallel \_))$. Then there must be an instance I of R such that $I = \{s, t\}$, $I \models \Sigma$ but $I \not\models (R : \overline{X} \rightarrow A, (\overline{t_p} \parallel \_))$. From the induction hypothesis we know that $s[\overline{X}^{(i-1)}] = t[\overline{X}^{(i-1)}]$. Let c_p be a tuple of constants such that c_p is equal to s (and hence also to t) on $\overline{X}_f^{(i-1)}$. This implies that I is a counterexample to $(R : [\overline{X}, \overline{X}_f^{(i-1)}] \rightarrow A, (c_p[\overline{X}_f^{(i-1)}] \bowtie \overline{t_p} \parallel \_))$ and therefore $A \notin \Sigma_v^*(\overline{X} \cup \overline{X}_f^{(i-1)}, c_p[\overline{X}_f^{(i-1)}] \bowtie \overline{t_p})$. Hence, A cannot be in $\overline{X}^{(i)}$ either, contradicting our assumption.

We use the following algorithm to compute $\overline{X}^{(n)}$. For each i, (i) execute v-Closure on input Σ_v, $\overline{X} \cup \overline{X}_f^{(i-1)}$ and $c_p[\overline{X}_f^{(i-1)}] \bowtie \overline{t_p}$, for all c_p's that are consistent with Σ and $\overline{t_p}$; and (ii) take the intersection of the result sets. The algorithm stops when it reaches $\overline{X}^{(n)} = \overline{X}^{(n+1)}$. It is clear that the algorithm runs in $O(|\mathsf{finattr}(R)||\mathsf{fdom}|^{|\mathsf{finattr}(R)|}|\Sigma|^2|\mathsf{attr}(\Sigma)|)$ time.

Based on the algorithm above, we extend the inductive proof given in cases **(1.b)** and **(2.a)** for the completeness of $\mathcal{I}$. That is, for all $A \in \overline{X}^{(n)}$, we construct a proof for $(R : \overline{X} \rightarrow A, (\overline{t_p} \parallel \_))$. First, in view of the completeness established in case **(2.a)** and the computation of $\overline{X}^{(i)}$ in terms of calls to v-Closure, we have for each $i \geq 0$ and each c_p, an $\mathcal{I}$-proof for the following:

$$(1)\ (R : [\overline{X}, \overline{X}_f^{(i)}] \rightarrow A_j, (c_p[\overline{X}_f^{(i+1)}] \bowtie \overline{t_p} \parallel \_)) \qquad \text{for each } A_j \in \overline{X}^{(i+1)};$$

Similar to case **(1.b)** we eliminate the dependency on c_p using FD4:

(2) $(R : \overline{X} \to A_j, (\overline{t_p} \parallel \_))$ for $A_j \in \overline{X}^{(1)}$; by (1) and FD4;
(3) $(R : [\overline{X}, \overline{X}_f^{(1)}] \to A_j, (\_, \ldots, \_, \overline{t_p} \parallel \_))$ for $A_j \in \overline{X}^{(2)}$; by (1) and FD4;
$\ldots$ $\ldots$
$(n+1)$ $(R : [\overline{X}, \overline{X}_f^{(n-1)}] \to A_j, (\_, \ldots, \_, \overline{t_p} \parallel \_))$ for $A_j \in \overline{X}^{(n)}$; by (1) and FD4;

Finally, we repeatedly apply the transitivity rule FD2:

$(n+2)$ $(R : \overline{X} \to A_j, (\overline{t_p} \parallel \_))$ for each $A_j \in \overline{X}^{(n)}$; and by (2), …,(n+1) and FD2.

In particular, given that $\Sigma \models (R : \overline{X} \to A, (\overline{t_p} \parallel \_))$ and hence $A \in \overline{X}^{(n)}$, we have that $\Sigma \vdash_{\mathcal{I}} (R : \overline{X} \to A, (\overline{t_p} \parallel \_))$, as desired. □

From these one can see that due to the richer semantics of CFDs, $\mathcal{I}$ is more complicated than Armstrong's Axioms. It is thus not surprising that the implication analysis of CFDs is more intriguing than their standard FD counterpart. Indeed, the theorem below shows that the implication problem for CFDs is intractable.

THEOREM 4.3. *The implication problem for* CFD*s is co*NP*-complete.*

PROOF. Consider a set Σ of CFDs and another CFD φ on a relation schema R, where Σ is consistent. The problem for determining whether or not $\Sigma \models \varphi$ is equivalent to the complement of the problem for determining whether or not there exists a nonempty instance I of R such that $I \models \Sigma$ and $I \models \neg\varphi$. Thus it suffices to show that the satisfiability problem for $\Sigma \cup \{\neg\varphi\}$ is NP-complete. Note that in contrast to the proof of Theorem 3.2, here we need to deal with $\neg\varphi$.

We first show that the satisfiability problem is in NP. Assume that $\varphi = (R : X \to Y, t_p)$. Similar to the upper bound proof of Theorem 3.2, it is easy to verify that if $\Sigma \cup \{\neg\varphi\}$ is satisfiable, then there exists an instance I consisting of two tuples s, t, such that $I \models \Sigma$, $s[X] = t[X] \asymp t_p[X]$, but either $s[Y] \neq t[Y]$, or $s[Y] \not\asymp t_p[Y]$ (resp. $t[Y] \not\asymp t_p[Y]$); note that here for each attribute $A \in X \cup Y$, $s[A] \in \mathsf{adom}(A)$ and $t[A] \in \mathsf{adom}(A)$. Then an NP algorithm similar to the one given in the proof of Theorem 3.2 suffices to check whether not $\Sigma \cup \{\neg\varphi\}$ is satisfiable.

We next show that the satisfiability analysis of $\Sigma \cup \{\neg\varphi\}$ is NP-hard by reduction from the non-tautology problem. The proof is similar to the lower bound proof of Theorem 3.2. Recall the statement of the non-tautology problem from that proof. Given an instance ϕ of the non-tautology problem, we define the same relation schema R and the same set Σ of CFDs on R as given in that proof. Furthermore, we define a single CFD $\varphi = (C \to C, (0 \parallel 1))$, which encodes that ϕ is false.

To show that the encoding is indeed a reduction, note that if ϕ is not a tautology, then it is already shown by the proof of Theorem 3.2 that there exists a nonempty instance I of R that satisfies Σ. Moreover, by the definition of Σ for any tuple t in I, $t[C] = 0$. Thus $I \not\models \varphi$, *i.e.,* $I \models \neg\varphi$. Conversely, suppose that there exists an instance I that satisfies Σ but $I \models \neg\varphi$. Then by $I \models \Sigma$ alone, the proof of Theorem 3.2 already shows that ϕ is not a tautology.

Putting this together, we have that the satisfiability problem is NP-complete and as a result, that its complement, the implication problem, is coNP-complete. □

Input: A set Σ of CFDs.
Output: A minimal cover of Σ.

1. **if** Σ is not consistent
2. **then return** $\emptyset$;
3. **for** each CFD $\varphi = (R : X \to A,\ t_p) \in \Sigma$
4. **for** each attribute $B \in X$
5. **if** $\Sigma \models (R : X \setminus \{B\} \to A,\ (t_p[X \setminus \{B\}] \parallel t_p[A]))$
6. **then** $\Sigma := \Sigma \setminus \{\varphi\} \cup \{(R : X \setminus \{B\} \to A,\ (t_p[X \setminus \{B\}] \parallel t_p[A]))\}$;
7. *mincover* $:= \Sigma$;
8. **for** each CFD $\varphi = (R : X \to A,\ t_p) \in \Sigma$
9. **if** $\Sigma \setminus \{\varphi\} \models \varphi$
10. **then** remove φ from *mincover*;
11. **return** *mincover*;

Fig. 5. Algorithm MinCover

The good news is that when the relation schema is predefined as commonly found in data cleaning applications, the implication analysis of CFDs can be conducted efficiently, as stated by the next result.

Corollary 4.4. *Given a set Σ of CFDs and a single CFD φ defined on a schema R, whether or not $\Sigma \models \varphi$ can be decided in $O(|\Sigma|^2|\mathsf{attr}(\Sigma)|)$ time if no attributes in $\mathsf{attr}(\Sigma)$ have a finite domain; and it is in $O(|\Sigma|^2)$ time if the schema is fixed.*

Proof. This follows immediately from the complexity analyses of the algorithms for deciding $\Sigma \models \varphi$ given in the proof of Theorem 4.2. □

4.2 Computing Minimal Covers of CFDs

As an application of consistency and implication analyses of CFDs, we present an algorithm for computing a minimal cover Σ_{mc} of a set Σ of CFDs. The cover Σ_{mc} is equivalent to Σ but does not contain redundancies, and thus is often smaller than Σ. Since the costs of checking and repairing CFDs are dominated by the size of the CFDs to be checked along with the size of the relational data, a non-redundant and smaller Σ_{mc} typically leads to less validating and repairing costs. Thus finding a minimal cover of input CFDs serves as an optimization strategy for data cleaning.

Following its traditional-FD counterpart (see, *e.g.,* [Abiteboul et al. 1995]), we define a *minimal cover* Σ_{mc} of a set Σ of CFDs to be a set of CFDs such that (1) $\Sigma_{\mathrm{mc}} \equiv \Sigma$, *i.e.,* Σ_{mc} and Σ are equivalent; (2) no proper subset of Σ_{mc} implies Σ_{mc}, *i.e.,* Σ_{mc} is nonredundant; and (3) each CFD in Σ_{mc} is of the form $(R : X \to A, t_p)$ as mentioned earlier, and moreover, for each $(R : X \to A,\ t_p)$ in Σ_{mc}, there exists no $(R : X' \to A,\ t_p[X' \cup A])$ in Σ_{mc} such that $X \subset X'$, *i.e.,* Σ_{mc} is canonical. Intuitively, conditions (1) and (2) assure that Σ_{mc} contains no redundant CFDs, and condition (3) ensures that Σ_{mc} does not have redundant attributes or patterns.

Example 4.5. Let Σ consist of $\psi_1 = (R : A \to B, (\_ \parallel b))$, $\psi_2 = (R : B \to C, (\_ \parallel c))$ and $\varphi = (R : A \to C, (a \parallel \_))$. A minimal cover Σ_{mc} of Σ consists of $\psi_1' = (\emptyset \to B, (b))$ and $\psi_2' = (\emptyset \to C, (c))$. This is because (1) $\{\psi_1, \psi_2\} \models \varphi$, which can be easily verified; (2) ψ_1 can be simplified to ψ_1' by removing the redundant attribute A (by the rule FD3 in $\mathcal{I}$), and (3) similarly, ψ_2 can be simplified to ψ_2'.

$Q^C_{\varphi_2}$ **select** * **from** cust t, T_2 t_p
where t[CC] $\asymp$ t_p[CC] **and** t[AC] $\asymp$ t_p[AC] **and** t[PN] $\asymp$ t_p[PN] **and**
(t[STR] $\not\asymp$ t_p[STR] **or** t[CT] $\not\asymp$ t_p[CT] **or** t[ZIP] $\not\asymp$ t_p[ZIP])

$Q^V_{\varphi_2}$ **select distinct** CC, AC, PN **from** cust t, T_2 t_p
where t[CC] $\asymp$ t_p[CC] **and** t[AC] $\asymp$ t_p[AC] **and** t[PN] $\asymp$ t_p[PN] **and**
(t_p[STR] = '_' **or** t_p[CT] = '_' **or** t_p[ZIP] = '_')
group by CC, AC, PN **having count** (**distinct** STR, CT, ZIP)> 1

Fig. 6. SQL queries for checking CFD φ_2

We give an algorithm, MinCover, for computing a minimal cover in Fig. 5. It is an extension of its standard FD counterpart [Maier 1980]. First, MinCover checks whether or not Σ is consistent (lines 1-2). If Σ is consistent, it proceeds to remove redundant attributes in the CFDs of Σ (lines 3–6). We use $(t_p[X \setminus \{B\}] \parallel t_p[A])$ to denote the pattern tuple t'_p such that $t'_p[A] = t_p[A]$ and $t'_p[C] = t_p[C]$ for each $C \in X \setminus \{B\}$. Next, it removes redundant CFDs from Σ (lines 8–10). From Proposition 3.5 and Corollary 4.4 it follows that MinCover is able to compute a minimal cover efficiently when the schema is predefined, in $O(|\Sigma|^3)$ time.

5. DETECTING CFD VIOLATIONS

A first step for data cleaning is the efficient detection of constraint violations in the data. In this section we develop techniques to detect violations of CFDs. Given an instance I of a relation schema R and a set Σ of CFDs on R, it is to find all the *violating tuples* in I, *i.e.,* the tuples that (perhaps together with other tuples in I) violate some CFD in Σ. We first provide an SQL technique for finding violations of a single CFD, and then generalize it to validate multiple CFDs. Finally we present an incremental technique for validating CFDs. It is desirable to use *just* SQL to find violations: this makes detection feasible in any standard relational DBMS without requiring any additional functionality on its behalf.

5.1 Checking a Single CFD with SQL

Consider a CFD $\varphi = (R : X \to A,\ T_p)$. For simplicity, we assume that the right-hand side of a CFD consists of a single attribute only. Our solutions can be trivially extended to multiple attributes, as illustrated by examples. Given the CFD φ, the following two SQL queries suffice to find the tuples violating φ:

Q^C_φ **select** * **from** R t, T_p t_p
where $t[X] \asymp t_p[X]$ **and** $t[A] \not\asymp t_p[A]$

Q^V_φ **select distinct** X **from** R t, T_p t_p
where $t[X] \asymp t_p[X]$ **and** $t_p[A]$ = '_'
group by X **having count** (**distinct** A)> 1

where for an attribute $B \in (X \cup A)$, $t[B] \asymp t_p[B]$ is a short-hand for the SQL expression ($t[B] = t_p[B]$ **or** $t_p[B]$ = '_'), while $t[B] \not\asymp t_p[B]$ is a short-hand for ($t[B] \neq t_p[B]$ **and** $t_p[B] \neq$ '_').

Intuitively, detection is a two-step process, each conducted by a query. Initially, query Q^C_φ detects *single-tuple* violations, *i.e.,* the tuples t in I that match some pattern tuple $t_p \in T_p$ on the X attributes, but t does not match t_p in A since the *constant* value $t_p[A]$ is different from $t[A]$. That is, Q^C_φ finds violating tuples based on differences in the constants in the tuples and T_p patterns.

On the other hand, query Q^V_φ finds *multi-tuple* violations, *i.e.,* tuples t in I for which there exists a tuple t' in I such that $t[X] = t'[X]$ and moreover, both t and t' match a pattern t_p on the X attributes, value $t_p[A]$ is a variable, but $t[A] \neq t'[A]$. Query Q^V_φ uses the **group by** clause to group tuples with the same value on X and it counts the number of distinct instantiations in $t_p[A]$. If there is more than one instantiation, then there is a violation. Note that Q^V_φ returns only the X attributes of violating tuples, to make the output more concise, since the complete tuples can be easily obtained using an additional SQL query.

Example 5.1. Recall CFD φ_2 given in Fig. 2. Over a cust instance I, the SQL queries $Q^C_{\varphi_2}$ and $Q^V_{\varphi_2}$ shown in Fig. 6 determine whether or not I satisfies φ_2. Executing these queries over the instance of Fig. 1, it returns tuples t_1, t_2 (due to $Q^C_{\varphi_2}$), and t_3 and t_4 (due to $Q^V_{\varphi_2}$).

A salient feature of our SQL translation is that tableau T_p is treated an ordinary *data table*. Therefore, each query is bounded by the size of the embedded FD $X \rightarrow A$ in the CFD, and is *independent* of the (possibly large) tableau T_p.

5.2 Validating Multiple CFDs

A naive way to validate a set Σ of CFDs is to use one query pair for each CFD φ in Σ. This approach requires $2 \times |\Sigma|$ passes of the underlying relation. We next present an alternative approach that only requires two passes. The key idea is to generate a *single* query pair to check all the constrains in Σ. The proposed solution works in two phases. In its first phase, it performs a linear scan of all the pattern tableaux belonging to CFDs in Σ and *merges* them, generating a single tableau called T_Σ. Intuitively, tableau T_Σ is such that it captures the constraints expressed by all the tableaux of the CFDs in Σ. Then in its second phase, it generates a query pair that finds tuples violating CFDs in Σ.

5.2.1 *Merging Multiple CFDs.* Consider a set Σ which, *w.l.o.g.*, contains just two CFDs φ and φ' on R, where $\varphi = (R : X \rightarrow A,\ T)$ and $\varphi' = (R : X' \rightarrow A',\ T')$. For now, assume that neither A nor A' belong to $X \cup X'$. We remove this assumption later. There are two main challenges for the generation of the merged tableau T_Σ. The first challenge is that tableaux T and T' may not be *union-compatible*, *i.e.,* $X \neq X'$ or $A \neq A'$. We thus need to extend tableau T (resp. T') with all the attributes in $(X \cup A) - (X' \cup A')$ (resp. $(X' \cup A') - (X \cup A)$ for T'). For each attribute B in $(X \cup A) - (X' \cup A')$ and each tuple t_p in the original tableau T, we set the value of $t_p[B]$ to be a *special symbol* denoted by '@', which denotes intuitively a *don't care* value. After this extension, the resulted tableaux are union-compatible. Then tableau T_Σ is defined to be their union. Figure 7 shows how the CFDs φ_2 and φ_3 of Fig. 2 can be made union-compatible.

Given the presence of '@', we need to reformulate CFD satisfaction. Let $Z = X \cup X'$ and $W = A \cup A'$. Consider a tuple $t_p[Z, W]$ in tableau T_Σ which includes '@'. We use $Z^{free}_{t_p}$ and $W^{free}_{t_p}$ to denote the subset of Z and W attributes of t_p that is '@'-free, *i.e.,* it has no '@' symbol. A relation I of R *satisfies* the CFD φ_Σ whose tableau is T_Σ, denoted by $I \models \varphi_\Sigma$, if for *each pair* of tuples t_1, t_2 in the relation I, and for *each* tuple t_p in the pattern tableau T_Σ of φ_Σ, if $t_1[Z^{free}_{t_p}] = t_2[Z^{free}_{t_p}] \asymp t_p[Z^{free}_{t_p}]$ then $t_1[W^{free}_{t_p}] = t_2[W^{free}_{t_p}] \asymp t_p[W^{free}_{t_p}]$.

$\varphi_4 = ([\mathsf{CC}, \mathsf{AC}, \mathsf{PN}] \rightarrow [\mathsf{STR}, \mathsf{CT}, \mathsf{ZIP}],\ T_4)$, where T_4 is

CC	AC	PN	STR	CT	ZIP
_	_	_	_	_	_
01	908	_	_	MH	_
01	212	_	_	NYC	_
_	_	@	@	_	@
01	215	@	@	PHI	@

Fig. 7. Merging of CFDs φ_2 and φ_3

id	CC	AC	CT
1	_	_	@
2	01	215	@
3	44	131	@
4	@	@	_

(a) Tableau T_Σ^Z

id	CT	AC
1	_	@
2	PHI	@
3	GLA	@
4	@	_

(b) Tableau T_Σ^W

Fig. 8. T_Σ for CFDs φ_3 and φ_5

For the second challenge, consider the detection of violations of a single CFD using SQL. Note that when writing the SQL queries, we assume implicit knowledge of whether an attribute is part of the left-hand or right-hand side of the CFD. Now consider two simple CFDs on R: $\varphi = (R : A \rightarrow B, T)$ and $\varphi' = (R : B \rightarrow A, T')$. Suppose that we have made the tableaux of the CFDs union-compatible. One might want to take the union of these two tableaux to generate T_Σ. Note that we cannot directly use the method given in the previous section. Attribute A is in the left-hand side, for tuples coming from φ, while it is part of the right-hand side, for tuples coming from φ'. Thus it seems that we need to distinguish the two sets of tuples and treat each set separately, something that counters the benefits of CFD merging.

We address this by splitting the tableau T of each CFD $\varphi = (R : X \rightarrow A,\ T)$ into two parts, namely, T^X and T^A, one tableau for X and one for A attributes of φ. Then tableau T_Σ^Z (and similarly T_Σ^W) is generated by making all the T^X tableaux in Σ union-compatible (similarly for the T^A tableau). Note that an attribute can appear in both T_Σ^Z and T_Σ^W. To be able to restore pattern tuples from T_Σ^Z and T_Σ^W, we create a distinct *tuple id* t_p[id] for each pattern tuple t_p, and associates it with the corresponding tuples in T_Σ^Z and T_Σ^W. For example, consider CFD φ_3 shown in Fig. 2 and $\varphi_5 = (R : [\mathsf{CT}] \rightarrow [\mathsf{AC}], T_5)$, where T_5 consists of a single tuple $(\_ \parallel \_)$. Figure 8 shows their merged T_Σ^Z and T_Σ^W tableaux. Note that attributes CT and AC appear in both tableaux.

5.2.2 *Query Generation.* During the second phase of our approach, we generate a single pair of SQL queries for T_Σ. This query generation, however, introduces new challenges. Recall that query Q_φ^V for some CFD $\varphi = (R : X \rightarrow A,\ T)$ requires a **group by** clause over all the X attributes. Now consider tableau T_Σ^Z in Fig. 8. It is not hard to see that if we use the **group by** clause over all the attributes in T_Σ^Z, we are not going to detect all (if any) violations since, for example, for the first three tuples in T_Σ^Z the '@' in attribute CT indicates that, while detecting violations, we should only group by the first two attributes and ignore the value of attribute CT. Similarly for the last tuple in T_Σ^Z, the '@' in attributes CC and AC indicates that

we should only consider the value of CT. The example suggests that our SQL query should change the set of **group by** attributes, based on the contents of each tuple. We next show how this can be achieved while still keeping the query size bounded by the size of the embedded FD $Z \rightarrow W$ in the merged tableau, and *independent* of the size of the merged tableau. Central to our approach is the use of the **case** clause of SQL (supported by popular DBMS like DB2, Oracle, MySQL).

Consider the merged tableaux T_Σ^Z and T_Σ^W from a set Σ of CFDs over a relation schema R and let I be an instance of R. Then the following two SQL queries can be used to detect tuples of I violating φ:

Q_Σ^C **select** * **from** $R\ t,\ T_\Sigma^Z\ t_p^Z,\ T_\Sigma^W\ t_p^W$
where $t_p^Z[\mathrm{id}] = t_p^W[\mathrm{id}]$ **and** $t[Z] \asymp t_p^Z[Z]$ **and** $t[W] \not\asymp t_p^W[W]$

Q_Σ^V **select distinct** Z **from** Macro t^M
group by Z **having count** (**distinct** W)> 1

where Macro is:

select (**case** $t_p^Z[B_i]$ **when** '@' **then** '@' **else** $t[B_i]$ **end**) **as** $B_i \cdots$
(**case** $t_p^W[C_j]$ **when** '@' **then** '@' **else** $t[C_j]$ **end**) **as** $C_j \cdots$
from $R\ t, T_\Sigma^Z\ t_p^Z, T_\Sigma^W\ t_p^W$
where $t_p^Z[\mathrm{id}] = t_p^W[\mathrm{id}]$ **and** $t[Z] \asymp t_p^Z[Z]$ **and** ($t_p^W[C_1] =$ '_' **or** $\cdots$ **or** $t_p^W[C_n] =$ '_')

Here for each attribute $B_i \in Z$, $t[B] \asymp t_p[B]$ now accounts for '@' and is a shorthand for ($t[B_i] = t_p[B_i]$ **or** $t_p[B_i] =$ '_' **or** $t_p[B_i] =$ '@'), and for each $C_j \in W$, $t[C_j] \not\asymp t_p[C_j]$ stands for ($t[C_j] \neq t_p[C_j]$ **and** $t_p[C_j] \neq$ '_' **and** $t_p[C_j] \neq$ '@').

Note that query Q_Σ^C is similar in spirit to the SQL query that checks for violations of constants between the relation and the tableau, for a single CFD. The only difference is that now the query has to account for the presence of the '@' symbol in the tableau. Now consider relation Macro which is of the same sort as T_Σ^Z and T_Σ^W (we rename attributes that appear in both tableaux so as not to appear twice). Relation Macro is essentially the join on Z of relation I with the result of the join on tuple id $t_p[\mathrm{id}]$ of the two tableaux. The value of each attribute, for each tuple t^M in Macro, is determined by the **case** clause. Indeed, for each attribute $B \in Z$, $t^M[B]$ is set to be '@' if $t_p^Z[B]$ is '@', and is $t[B]$ otherwise; similarly for each $C \in W$ and $t^M[C]$. Note that relation I is not joined on W with the tableaux. Thus if for some tuple t with $t[Z] \asymp t_p^Z[Z]$, there exists an attribute C with $t_p^W[C]$ a constant and $t[C] \neq t_p^W[C]$ (*i.e.,* t violates the merged tableau) then $t^M[C]$ is set to be $t[C]$. This creates no problems since this violating tuple is already detected by Q_Σ^C.

Intuitively, Macro considers each tuple in the tableau, and uses it as a *mask* over the tuples of the relation. If the tableau tuple indicates a *don't care* value for an attribute, all the (possibly different) attribute values in the relation tuples are masked and replaced by an '@' in Macro. Figure 9 shows the result of joining the fourth tuple of tableaux T_Σ^Z and T_Σ^W in Fig. 8 with the cust relation of Fig. 1. Note that the query masks the attributes values of CC and AC. This masking allows the subsequent **group by** over X to essentially consider, for each tuple, only the subset of Z that does not have any *don't care* values. Note that although $Z = \{\mathsf{CC}, \mathsf{AC}, \mathsf{CT}\}$, the **group by** by query Q_Σ^V essentially performs a **group by** over *only* attribute CT. The query returns the NYC tuples which violate φ_5.

In this way we generate a *single pair* of SQL queries to validate a *set* Σ of CFDs,

CC	AC	CT	CT′	AC′
@	@	NYC	@	908
@	@	NYC	@	212
@	@	PHI	@	215
@	@	EDI	@	131

Fig. 9. Macro relation instance

while guaranteeing that the queries are bounded by the size of the embedded FDs in Σ, *independent* of the size of the tableaux in Σ. Furthermore, to validate Σ only two passes of the database is required.

5.3 Incremental CFD Detection

Consider an instance I of a relation schema R. For simplicity, consider a single CFD $\varphi = (R : X \rightarrow A,\ T_p)$ (the incremental technique presented below can be extended to deal with multiple CFDs, along the same lines as Section 5.2). Given the methodology presented thus far, we can check for violations of φ by issuing the pair of queries Q^C_φ and Q^V_φ over I. An interesting questions is then what happens if the instance I changes? As tuples are inserted into or deleted from I, resulting a new instance I^{new}, a naive solution would be a batch approach that re-issues queries Q^C_φ and Q^V_φ over I^{new}, starting from scratch in response to updates, something that requires two passes of the entire instance each time the queries are re-issued.

Intuitively, however, one expects that a tuple insertion leaves a large portion of instance I unaffected when CFD violations are concerned. An inserted tuple t might introduce *new* violations, but only with tuples that are already in the instance and which match t in the X attributes. Therefore, it makes sense to *only* access these tuples and *only* detect the possible newly introduced violations due to the inserted tuple. Thus, *incremental* detection can potentially save a large number of disk accesses, since instead of performing two passes of the underlying data on each tuple insertion (naive method), we only need to access the tuples that match the inserted tuple t in the X attributes. Similarly in the case of deletion, by deleting a tuple t we might inadvertibly *repair* some of the violations in I that the deleted tuple was causing (again with tuples matching t in the X attributes). Therefore, it makes sense to *only* detect which of the existing violations concerning the deleted tuple are affected. The following example better illustrates the above.

Example 5.2. Recall from Example 5.1 that tuples t_1 to t_4 in Fig. 1 violate CFD φ_2. Now consider inserting the tuple t_7 : (01, 215, 3333333, Bill, Main Rd. PHI, 02394) in the relation of the figure. It is easy to check that tuples t_5 and t_7 violate φ_2 (due to $Q^V_{\varphi_2}$). Still, the newly inserted tuple does not affect the violations detected between the first four tuples. Note that an incremental detection would require that we only access tuple t_5, instead of the whole relation.

Now consider again the instance in Fig. 1 and assume that we delete tuple t_4 from it. Then tuple t_3 no longer violates φ_2 since the deletion of t_4 inadvertibly repaired the violation caused by tuples t_3 and t_4. Such a deletion only requires accessing tuple t_3 and does not affect the violation caused by tuples t_1 and t_2.

We next present a method to *incrementally* detect CFD violations, given a set of insertions and deletions to an instance I. Although the incremental method has

	CC	AC	PN	NM	STR	CT	ZIP	$\beta^C_{\varphi_2}$	$\beta^V_{\varphi_2}$
t_1:	01	908	1111111	Mike	Tree Ave.	NYC	07974	1	0
t_2:	01	908	1111111	Rick	Tree Ave.	NYC	07974	1	0
t_3:	01	212	2222222	Joe	Elm Str.	NYC	01202	0	1
t_4:	01	212	2222222	Jim	Elm Str.	NYC	02404	0	1
t_5:	01	215	3333333	Ben	Oak Ave.	PHI	02394	0	0
t_6:	44	131	4444444	Ian	High St.	EDI	EH4 1DT	0	0

Fig. 10. The cust relation instance with logging information

the same worst-case performance as the naive method (two passes of the underlying instance), its expected performance is that only a small number of tuples are accessed, which will be verified in the next section by our experiments.

5.3.1 *Logging of violations.* The incremental detection requires us to extend the schema R of an instance relation I to record which tuples violate which CFDs in a given set Σ. In more detail, for each CFD $\varphi \in \Sigma$ we add two Boolean attributes β^C_φ and β^V_φ to the schema of R. We use R^{log} to denote the new schema. For each tuple $t \in I$, we create a tuple t' in R^{log} such that $t'[\mathsf{attr}(R)] = t[\mathsf{attr}(R)]$. Furthermore, in attribute $t'[\beta^C_\varphi]$ (resp. $t'[\beta^V_\varphi]$) we record whether or not t violates CFD φ due to Q^C_φ (resp. Q^V_φ). Note that our logging mechanism imposes minimum overhead since for each tuple and each CFD only two additional bits are required.

We assume that, initially, we execute queries Q^C_φ and Q^V_φ and we use the result of the two queries to initialize the values of attributes β^C_φ and β^V_φ, through a simple SQL update statement of the following form for β^C_φ (similarly for β^V_φ):

U^C_φ **update** R^{log} t' **set** $t'[\beta^C_\varphi] = 1$
where $t'[\mathsf{attr}(R)]$ **in** (Q^C_φ)

One needs only to select all those tuples with both β^C_φ and β^V_φ equal to false, for each $\varphi \in \Sigma$, and then project on $\mathsf{attr}(R)$, in order to retrieve from R^{log} the tuples that do not violate any of the CFDs. Figure 10 shows the instance of Fig. 1 after its schema has been extended appropriately to log violations for CFD φ_2.

5.3.2 *Handling tuple deletions.* Consider a CFD $\varphi = (R : X \rightarrow A,\ T_p)$ and an instance I^{log} whose schema R^{log} includes attributes β^C_φ and β^V_φ. In response to deletion of tuple t from I^{log}, the incremental detection of violations has two steps.

Step 1: **delete from** R^{log} t' **where** $t' = t$

Step 2: **update** R^{log} t' **set** $t'[\beta^V_\varphi] = 0$
where $t'[\beta^V_\varphi] = 1$ **and** $t'[X] = t[X]$ **and**
$1 = (\ \textbf{select}\ \ \textbf{count}\ (\textbf{distinct}\ A)\ \ \textbf{from}\ R^{log}\ \ t''$
where $t''[X] = t[X])$

In more detail, the SQL query in the first step simply deletes from R^{log} the tuple corresponding to t. The second step checks for tuples that (a) violate φ ($t'[\beta^V_\varphi] = 1$), (b) have the same values on the X attributes with t, and (c) all these identified tuples have the same A attribute value. It is easy to see that if a set of tuples satisfies the above three conditions, then each of the tuples in the set violated φ only due to t. Since we delete t, each of the tuples now satisfies φ. Therefore, we set $t'[\beta^V_\varphi]$ to false. Note that a tuple deletion only affects violations that are caused

by the presence of '_' in the tableau, hence we focus only on the β_φ^V attribute. Also note that the above procedure need not access the pattern tableau T_p of φ, resulting in additional savings in terms of execution time.

Example 5.3. Consider the instance in Fig. 10 and assume that we delete tuple t_4. Then the second step of our incremental detection will select tuple t_3 and set $t_3[\beta_{\varphi_2}^V]$ to false since there is no other tuple in the instance that has the same values on the CC, AC and PN attributes as t_3 but differs from t_3 in STR, CT or ZIP. Hence tuple t_3 no longer violates φ_2. Note that our incremental detection, using appropriate indexes, only accesses tuple t_3. Contrast this with the non-incremental detection which requires to access the whole relation twice.

A question is what happens when we want to do *batch* deletion, *i.e.,* delete a set of tuples. Obviously, we could execute the above two steps once for each tuple in the set. We can actually do better than that since it suffices to execute the above steps once for each *distinct* value of X attributes that is deleted. So, if for example we delete both tuples t_3 and t_4 from the instance in Fig. 10, we only need to execute the two steps once since the two tuples have the same value on the X attributes.

5.3.3 *Handling tuple insertions.* Assume that we want to insert a tuple t into I^{log}. Then the incremental detection of violations has the following three steps.

Step 1: **insert into** R^{log} **values** t

Step 2: **update** R^{log} t' **set** $t'[\beta_\varphi^C] = 1$
where $t' = t$ **and**
exists (**select** * **from** T_p
where $t_p[X] \asymp t'[X]$ **and** $t_p[A] \not\asymp t'[A]$)

Step 3: **update** R^{log} t' **set** $t[\beta_\varphi^V] = 1$, $t'[\beta_\varphi^V] = 1$
where $t'[X] = t[X]$ **and** $t'[A] \neq t[A]$ **and**
exists (**select** * **from** T_p t_p
where $t_p[X] \asymp t[X]$ **and** $t_p[A] =$ '_')

The first step simply inserts the tuple t into relation R^{log}, where we assume that both the β_φ^C and β_φ^V attributes are set to false, for each newly inserted tuple. Similar to Q_φ^C, the second step checks for violations in the constants between the newly inserted tuple and the pattern tableau T_p. If such violations exist, it sets the value of β_φ^C in the inserted tuple to true. Similar to Q_φ^V, the final step checks for tuples that (a) have the same values on the X attributes with t, (b) differ from t on the A attribute, and (c) the inserted tuple t satisfies φ *by itself*. If these conditions are satisfied, each identified tuple and t, when put together, violate φ, and we set the value of the β_φ^V attribute of each such tuple to true. We slightly abuse notation in the last step to also set, with the same statement, the β_φ^V attribute of t to true.

We now consider batch insertions involving a set of tuples, say ΔI^{log}. Obviously, one might consider executing the above steps once for each tuple in ΔI^{log}. An alternative strategy is to treat ΔI^{log} as an independent instance whose tuples we need to merge with the ones in I^{log}. We distinguish five different steps here:

Step 1: **update** ΔR^{log} t' **set** $t'[\beta_\varphi^C] = 1$
where $t'[\mathsf{attr}(R)]$ **in** (Q_φ^C)

Step 2: **update** R^{log} t' **set** $t'[\beta_\varphi^V] = 1$

	CC	AC	PN	NM	STR	CT	ZIP	$\beta^C_{\varphi_2}$	$\beta^V_{\varphi_2}$
t_7:	01	212	5555555	Tim	Main Str.	CHI	01202	0	0
t_8:	01	212	2222222	Sam	Elm Str.	NYC	01202	0	0
t_9:	44	131	4444444	Al	King St.	EDI	EH4 1DT	0	0

Fig. 11. An instance ΔI^{log} used for batch insertion

where $t'[\beta^C_\varphi] = 0$ **and** $t'[\beta^V_\varphi] = 0$ **and**
exists (**select** * **from** T_p t_p
where $t_p[X] \asymp t[X]$ **and** $t_p[A] =$ '_') **and**
exists (**select** * **from** ΔR^{log} t''
where $t''[X] = t'[X]$ **and** $t''[A] \neq t'[A]$) **and**

Step 3: **update** ΔR^{log} t' **set** $t'[\beta^V_\varphi] = 1$
where **exists** (**select** * **from** R^{log} t''
where $t''[X] = t'[X]$ **and** $t''[\beta^V_\varphi] = 1$)

Step 4: **update** ΔR^{log} t' **set** $t'[\beta^V_\varphi] = 1$
where $t'[\beta^C_\varphi] = 0$ **and** $t'[\beta^V_\varphi] = 0$ **and**
$t'[X]$ **in** (**select** X **from** ΔR^{log} t'', T_p t_p
where $t''[\beta^C_\varphi] = 0$ **and** $t''[\beta^V_\varphi] = 0$ **and** $t''[X] \asymp t_p[X]$
$t_p[A] =$ '_'
group by X
having count (**distinct** A)> 1)

Step 5: **insert into** R^{log} **values** (**select** * **from** ΔR^{log})

where ΔR^{log} denotes the schema of ΔI^{log}, which is identical to R^{log}. During the first step, we focus on the newly inserted tuples and we identify which tuples independently violate φ due to Q^C_φ. This is an unavoidable step whose cost cannot be reduced since we have to consider each inserted tuple in isolation. However, by executing Q^C_φ only over ΔR^{log}, we avoid re-detecting such violations over R^{log}.

The second step looks for tuples in R^{log} that were clean before the insertion but will now violate φ, once the tuples in ΔR^{log} are inserted. The tuples in R^{log} that are affected by the insertion are such that they have the same values on the X attributes with some tuple in ΔR^{log} but their values differ on the A attribute.

The third step attempts to leverage the knowledge of violations in R^{log} in order to detect violations in ΔR^{log}. If a tuple t' in ΔR^{log} has the same values on the X attributes with some tuple t'' in R^{log} whose β^V_φ is true, then t' must also have β^V_φ set to true. This is because we already know for the tuples in R^{log} with specific values on the X attributes whether more than one values on the A attribute exist.

Finally, there is only one more case to consider, namely, whether there are any *clean* tuples in ΔR^{log} (with both β^C_φ and β^V_φ equal to false) that together with some other clean tuples in ΔR^{log} violate φ. The last step detects such tuples by checking whether any tuples have the same values on the X attributes but different values on the A attribute. For all the detected tuples, the value of β^V_φ is set to true.

In last step, we simply insert the tuples in ΔR^{log} into R^{log}.

Example 5.4. Consider the instance in Fig. 11 and assume that we want to insert its tuples into the instance in Fig. 10. Then the first step above will set $t_7[\beta^C_{\varphi_2}]$ to true, since the value of $t_7[\mathsf{CT}]$ is "CHI" instead of "NYC". The second step will set $t_6[\beta^V_{\varphi_2}]$ to true, since tuples t_6 and t_9 violate φ_2. The third step will set $t_9[\beta^V_{\varphi_2}]$ also

to true, while none of the remaining steps will alter any tuples.

6. EXPERIMENTAL STUDY

In this section we present our findings about the performance of our techniques for (incrementally) detecting CFD violations over a variety of data sizes, and number and complexity of CFDs. We distinguish four sets of experiments. After identifying a number of parameters that influence the detection of violations, in the first set of experiments we vary these parameters, and investigate the effects of each parameter combination on the execution time of the SQL detection queries. In the second experiments, we focus on the detection of multiple CFDs and study the benefits of merging multiple CFDs in a single tableau. In the first two sets of experiments we *only* report the time to execute the SQL detection queries and omit the time to report (or *mark*) the violating tuples. This omission does not affect the validity of our results since, for the first two experiments, marking the violating tuples only adds a constant to each reported time of each figure. In the third set of experiments we compare the effectiveness of CFDs in detecting dirty tuples versus its FD counterpart, as well as their running time. In the fourth set of experiments we focus on incremental detection and its benefits *w.r.t.* the non-incremental one. In the last two sets of experiments we report the sum of the time to execute the SQL detection query plus the time to mark the violating tuples.

6.1 Experimental setup

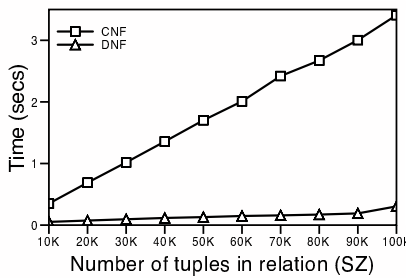
– **Hardware:** For the experiments, we used DB2 on an Apple Xserve with 2.3GHz PowerPC dual CPU and 4GB of RAM.

– **Data:** Our experiments used an extension of the relation in Fig. 1. Specifically, the relation models individual's tax-records and includes 8 additional attributes, namely, the state ST where a person resides, her marital status MR, whether she has dependents CH, her salary SA, tax rate TX on her salary, and 3 attributes recording tax exemptions, based on marital status and the existence of dependents.

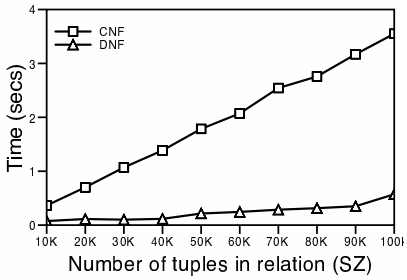
To populate the relation we collected real-life data: the zip and area codes for major cities and towns for all US states. Further, we collected the tax rates, tax and income brackets, and exemptions for each state. Using these data, we wrote a program that generates synthetic tax records.

We vary two parameters of the data instance in our experiments, denoted by SZ and NOISE. SZ determines the tuple number in the tax-records relation and NOISE the percentage of dirty tuples. As the data is generated, with probability NOISE, an attribute on the RHS of a CFD is changed from a correct to incorrect value (*e.g.,* a tax record for a NYC resident with a Chicago area code).

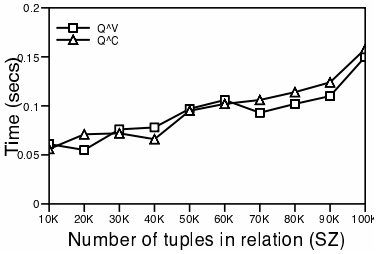
– **CFDs:** We used CFDs to model real-world semantics such as (a) zip codes determine states, (b) zip and cities determine states, and (c) states and salary brackets determine tax rates (a tax rate depends on both the state and employee salary), etc. We varied our CFDs using the following parameters: NUMCFDs determines the number of CFDs considered in an experimental setup, NUMATTRs the (max) attribute number in the CFDs, TABSZ the (max) tuple number in the CFDs, and NUMCONSTs the percentage of tuples with constants vs. tuples with variables in each CFD.



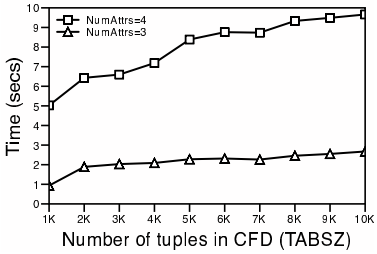
(a) CNF vs DNF (NUMCONSTs = 100%)



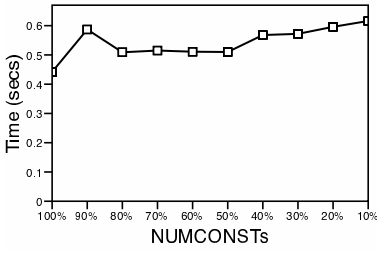
(b) CNF vs DNF (NUMCONSTs = 50%)



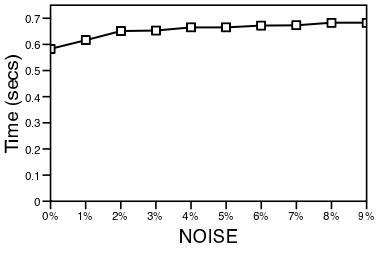
(c) Q^C_φ vs. Q^V_φ



(d) Scalability in TABSZ



(e) Scalability in NUMCONSTs



(f) Scalability in NOISE

Fig. 12. Experimental results

6.2 Detecting CFD violations

There are two alternative evaluation strategies for the SQL detection queries of Section 5. Key distinction between these two strategies is how we evaluate the **where** clause in each detection query. Specifically, note that the **where** clause of our SQL detection queries is in the conjunctive normal form (CNF). It is known that database systems do not efficiently execute queries in CNF since the presence of the OR operator leads the optimizer to select inefficient plans that do not leverage the available indexes. A solution to this problem is to convert conditions in the **where** clause into the disjunctive normal form (DNF). This conversion might cause an exponential blow-up in the number of conjuncts, but in this case, the blow-up is *w.r.t.* the number of attributes in the CFD, which is usually very small.

– CNF **vs.** DNF**:** In this experiment, we considered both evaluation strategies, under various settings, to determine the most efficient one. In more detail, we considered

relations with SZ from 10K to 100K tuples in 10K increments, and 5% NOISE. We considered two *representative* CFDs, each with NUMATTRs 3, where the first CFD had NUMCONSTs 100% (tuples with only constant) while the second had NUMCONSTs 50% (half the tuples had variables). In terms of CFD size, we set TABSZ to 1K (note that *each tuple* in the CFDs is a constraint itself). Figures 12(a) and 12(b) show the evaluation times for both evaluation strategies, for each of the two CFDs. As both graphs show, irrespective of data size and the presence of constants or variables, the DNF strategy clearly out-performs the CNF one. Furthermore, the figures illustrate the scalability of our detection queries SZ.

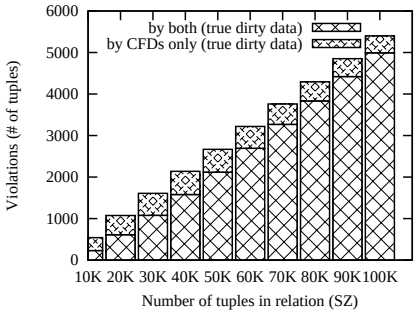
– Q_φ^C **vs.** Q_φ^V**:** We investigated how the detection time is split between the Q_φ^C and Q_φ^V queries. We considered relations with SZ from 10K to 100K tuples in 10K increments, and 5% NOISE. For the CFD, we consider one with NUMATTRs equal to 3, TABSZ to 1K and NUMCONSTs 100% (we had similar results for other values of NUMCONSTs). Figure 12(c) shows the evaluation times for each query in isolation and shows that both queries have similar loads and follow the same execution trend.

– **Scalability in** TABSZ**:** We studied the scalability of the detection queries with respect to TABSZ. In more detail, we fixed SZ to 500K with 5% NOISE. We considered two CFDs whose sizes varied from 1K to 10K, in 1K increments. The NUMATTRs was 3 for the first, and 4 for the second CFD considered. For all CFDs, NUMCONSTs was 50%. Figure 12(d) shows the detection times for the 2 CFDs. As is obvious from the figure, TABSZ has little impact on the detection times and dominant factors here are (a) the size of the relation, which is much larger than the tableaux, and (b) the number of attributes in the tableau, since these result in more complicated join conditions in the detection queries.

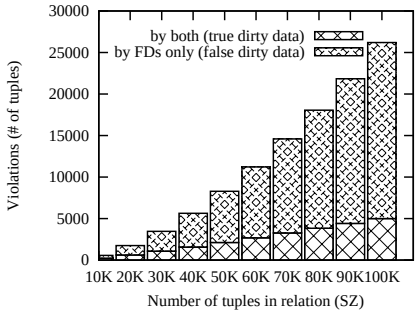
– **Scalability in** NUMCONSTs**:** We studied the impact of variables on the detection times. Fixing a relation with SZ 100K and NOISE 5% and a CFDs with TABSZ 1K, and NUMATTRs = 3, we varied NUMCONSTs between 100% (all constants) and 10% and measured the detection times over the relation. Figure 12(e) shows that variables do affect detection times and (not shown in the figure) moreover, as we increased both the percentage of variables and the number of attributes with variables, detection times increased noticeably. This is apparent, given that variables restrict the use of indexes while joining the relation with the tableau.

– **Scalability in** NOISE**:** We varied NOISE between 0% and 9% in a relation with SZ 100K, and considered a CFD with TABSZ 30K (we used all possible zip to state pairs, so as not to miss a violation), NUMATTRs 2, and NUMCONSTs 100%. As shown in Fig. 12(f), the level of NOISE has negligible effects on detection times.

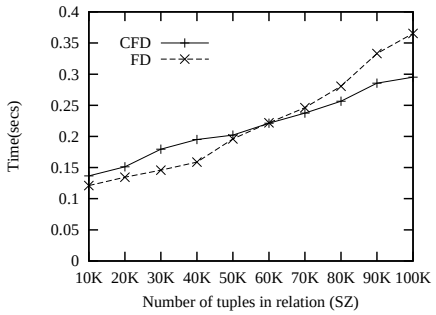
– **On the representation of variables:** One practical consideration was the representation of the unnamed variables '_'. Initially, we experimented by actually using the character '_' as an attribute value, to code variables. The performance of the SQL detection queries was satisfactory but we noticed that as the number of variables in a CFD increased, there was a corresponding increase in the detection times (as was already shown in Fig. 12(e)). As an alternative, we considered using the null value in a CFD to represent variables. Our SQL detection queries were affected since the term ($t_p[X_i]$ = '_') was now changed to ($t_p[X_i]$ **is** null). Performance-wise, there was a 10%-20% improvement on detection times. Although an increase on



(a) The violations detected by CFDs



(b) The violations detected by FDs



(c) Performance

Fig. 13. CFDs versus FDs

the number of variables in a CFD still resulted in an increase on times, detection times scaled much more gracefully with null than when the '_' character was used.

6.3 Comparing CFDs and FDs

We now compare the effectiveness and efficiency of CFDs in inconsistency detection, versus their FD counterpart, in terms of both detection time and the quality of dirty data that they detect. In this set of experiments, we used CFDs with all constant pattern tuples (NUMCONSTs = 100%) while for the FDs NUMCONSTs = 0%. In terms of the relation, we varied SZ from 10K to 100K tuples while NOISE was fixed at 5%. We set NUMATTRs to 2 and TABSZ to 33K. We kept a copy of clean data before adding any noise. In this way we can tell whether a tuple detected by CFDs (resp. FDs) is a *true* dirty tuple or a *false* dirty tuple.

Figure 13(a) shows the number of true dirty tuples detected by both CFDs and FDs vs. the number of those detected by CFDs alone. Clearly FDs missed a number of true dirty tuples that were caught by CFDs because a single true dirty tuple does not violate an FD while it may violate a CFD. From Fig. 13(a) it is clear that when the dataset gets larger, the chance is higher that a dirty tuple may incur conflict with some other tuples, and as a result, can be caught by FDs.

Figure 13(b) gives the number of tuples that FDs detected as dirty vs. the number of those detected by both CFDs and FDs. Clearly FDs marked a large number of *false* dirty tuples, because given a pair of tuples that violate an FD, the FD cannot distinguish which of the two tuples is dirty and which is clean. Therefore, the FD

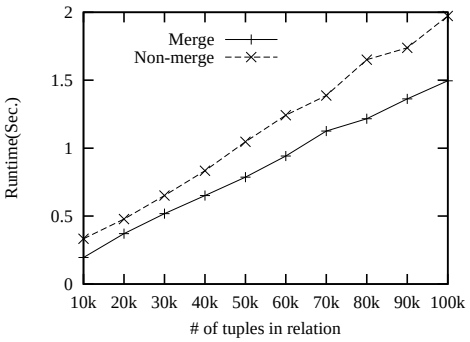


Fig. 14. Merging CFDs

returns both tuples as dirty. On the other hand, a CFD with constants at RHS is able to distinguish between the two. The figure shows that rate of false tuples caught by FDs grows with the data size.

Finally, Fig. 13(c) reports the detection time of using FDs vs. CFDs. As expected, since the size of the pattern tableaux in the CFDs is larger than that of FDs (all pattern tuples are wildcards), the CFD detection is slower on small-sized instances. However, when the instance size grows, the higher selectivity of the pattern tableaux in the CFDs makes CFD detection faster than FD detection. Moreover, for smaller TABSZ, our experiments (not shown) demonstrated an earlier crossing point (less than 20K tuples) and bigger performance gain (half time for 100K tuples).

6.4 Detecting Multiple CFD violations

In general, the performance of the merged scheme is hampered by the difficulty faced by optimizers when handling **where** clauses in CNF. The conversion to DNF is not an option here, because each disjunct in CNF consists of 3 terms, and thus the translation of CNF to DNF results in a **where** clause with 3^k conjuncts, where k is the number of attributes in the CFD. In practice this is much worse than the 2^k increase that results from translating Q^C_φ or Q^V_φ into DNF.

Given the blowup, and lacking an efficient optimizer for CNF queries, we devised with an *implementation* strategy for the merged scheme. In more detail, we reduced the number of disjuncts in the representation of $t[B_i] \asymp t_p[B_i]$ from three to just two. This was done by replacing in T^Z_Σ (and similarly in T^W_Σ) all occurrences of the '_' and "@" with the null symbol. Given a null value for some attribute $t^Z_p[B_i]$ in T^Z_Σ, we needed to distinguish whether the null stands for a '_' or "@". To this end, we introduced an additional Boolean column β^N_i for each attribute B_i in T^Z_Σ. The value of $t^Z_p[\beta^N_i]$ was set to true if the null stands for a '_', and to false if the null stands for a "@". Given this encoding, $t[B_i] \asymp t_p[B_i]$ now accounts for ($t[B_i]$ $= t_p[B_i]$ **or** $t_p[B_i]$ **is** null), *i.e.,* it only has two terms. We also had to change the **case** statement in Macro, to account for the new Boolean attribute β^N_i, as follows:

(**case** $t^Z_p[\beta^N_i]$ **when** 0 **then** "@" **else** $t[B_i]$ **end**) **as** B_i

Figure 14 shows the sum of detection times for two CFDs: $\varphi = (R : [\mathsf{AC}, \mathsf{CT}] \rightarrow [\mathsf{ST}],\ T_p)$ and $\varphi' = (R : [\mathsf{AC}] \rightarrow [\mathsf{ST}],\ T'_p)$, with TABSZ 500 and 200, respectively. The underlying relation has SZ that ranges from 10K to 100K, in 10K increments,

and NOISE 5%. The merged CFD has TABSZ 700 and its detection time is plotted. The benefit of merged CFD is noticeable versus the two individual CFDs. Our experiments indicate that CFD merging is mainly beneficial for highly-related CFDs, *i.e.,* CFDs for which there is a substantial overlap on the attributes they involve.

6.5 Incremental detection

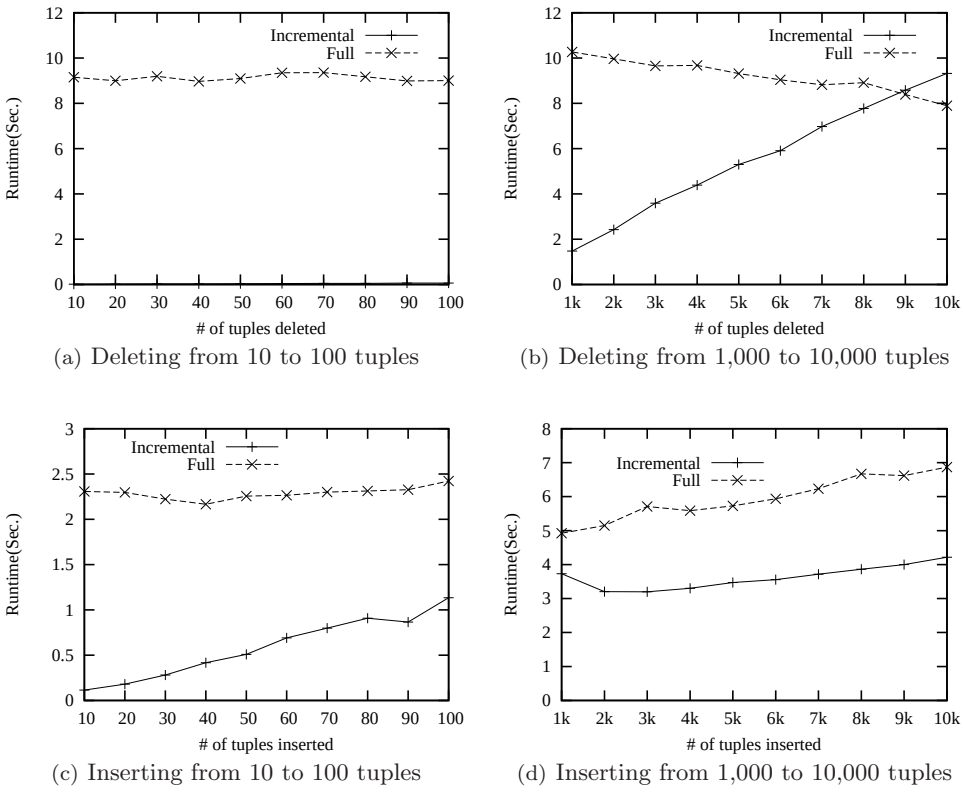
For our third set of experiments, we used a relation with SZ 100K, and NOISE 5%. In terms of the CFD, TABSZ was 500 and NUMATTRs was 3.

– **Single tuple deletions:** In this experiment, we consider sets of tuples ranging from 10 to 100 tuples, in 10 tuple increments. For each set, we delete its tuples one-by-one, and after each deletion we use incremental detection to discover violations. So, for the set of 10 tuples, we call incremental detection 10 times, once for each deleted tuple. In Fig. 15(a), we report the cumulative time of incremental detection, after all the tuples in the set have been deleted, *i.e.,* the reported time is the sum of running 10 times the incremental detection. At the same tine, we also report the time for full (non-incremental) detection, where full detection is performed only *once*, after all the tuples in the set have been deleted. So, while incremental detection is called after each tuple deletion, full detection is only called at the end of deleting all tuples. Note in the figure that the line for incremental detection falls on the x-axis and is not visible. On the other hand, full detection is an order of a magnitude slower, proving clearly the gains of the former method over the latter.

– **Batch tuple deletions:** We now consider deleting sets of tuples ranging from 1,000 to 10,000 tuples, in 1,000 increments. For each set, we perform batch deletion, use incremental detection, and measure its running time. Similarly, after each batch deletion we also use full detection, and measure its running time also. In Fig. 15(b), we report the measured times, for each set. As expected, the more tuples we delete, the faster full detection becomes since it has to consider less tuples after the deletion. At the same time, the larger the batch of tuples we delete, the more time incremental detection takes. This is because in Step 2 of the incremental detection during deletion, we have to consider an increasing number of tuples to incrementally detect. The crossing point is around 9,000 tuples. Given our initial relation of 100K tuples, even if we delete around 10% of this relation, a considerable portion by any standard, incremental detection is still a better choice than doing a full detection. In general one can achieve optimal detection times through a simple algorithm that considers the size of the base relation and the number of tuples to be deleted and it chooses between executing an incremental or a full detection.

– **Single tuple insertions:** Similar to single tuple deletions, we consider inserting sets of tuples ranging from 10 to 100 tuples, in 10 tuple increments. We also used the same experimental strategy as single tuple deletions by measuring the cumulative incremental detection time, for all the tuples in the set, versus the full detection time after all the tuples have been deleted. Figure 15(c) shows that for any reasonable number of insertions, incremental detection does better than periodically doing full detection. In the worst case, incremental detection is twice as fast as full detection, while for a few tuples it is almost an order of magnitude faster.

– **Batch tuple insertions:** Similar to batch tuple deletions, we consider batch



(a) Deleting from 10 to 100 tuples
(b) Deleting from 1,000 to 10,000 tuples
(c) Inserting from 10 to 100 tuples
(d) Inserting from 1,000 to 10,000 tuples

Fig. 15. Experimental results

tuple insertions. As expected, Fig. 15(d) shows that as the number of tuples inserted increases, so does the time to execute a full detection. A similar increase is noticed in the multi-step incremental detection and unlike batch deletions, there is no crossing point here even when a considerable number of new tuples are inserted. So, incremental detection is a clear winner, for batch tuple insertions.

7. RELATED WORK

The authors introduced CFDs in earlier work [Bohannon et al. 2007]. This paper extends [Bohannon et al. 2007] by including (a) proofs for all the theorems; some of the proofs are nontrivial and the techniques are interesting in their own right; (b) an algorithm for checking the consistency of a set of CFDs, which is in PTIME when either the database schema is predefined or no attributes involved in the CFDs have a finite domain (Section 3.1); (c) an approximation factor preserving reduction to MAXGSAT (Section 3.2); (d) a more concise sound and complete inference system with less axioms (Section 4); (e) incremental techniques for detecting CFD violations in response to changes to the underlying database (Section 5.3); and (f) an extensive experimental study compared to the preliminary study of [Bohannon et al. 2007] (Section 6).

We next discuss previous work on data cleaning and compare the study of CFDs

with related work on various extensions of FDs.

Related work on data cleaning. Prior work on constraint-based data cleaning has mostly focused on two topics, both introduced in [Arenas et al. 2003]: *repairing* is to find another database that is consistent and minimally differs from the original database (*e.g.,* [Arenas et al. 2003; Chomicki and Marcinkowski 2005a; Franconi et al. 2001]); and *consistent query answering* is to find an answer to a given query in every repair of the original database (*e.g.,* [Arenas et al. 2003; Wijsen 2005]).

Most earlier work (except [Franconi et al. 2001; Wijsen 2005]) considers traditional full (subsuming functional) dependencies and denial constraints. In these settings, complexity results [Arenas et al. 2003; Cali et al. 2003a; Chomicki and Marcinkowski 2005a; Greco et al. 2003; Wijsen 2005], algorithms [Arenas et al. 2003; Cali et al. 2003a; 2003b; Chomicki and Marcinkowski 2005a; Wijsen 2005], constraint rewriting techniques [Greco et al. 2003], representations of all repairs with logic programming [Bravo and Bertossi 2003; Franconi et al. 2001] or tableau [Wijsen 2005], and constraint repair based on techniques from model-based diagnosis [Gertz and Lipeck 1995] were developed, for single databases [Arenas et al. 2003; Cali et al. 2003a; Chomicki and Marcinkowski 2005a; Franconi et al. 2001; Greco et al. 2003; Wijsen 2005] and integration systems [Bravo and Bertossi 2003; Cali et al. 2003b; Greco et al. 2003] (see recent surveys on consistent query answering [Bertossi and Chomicki 2003] and on constraint repair [Chomicki and Marcinkowski 2005b]). As remarked earlier, full and denial constraints differ from CFDs in that they do not allow patterns with data values and cannot detect inconsistencies in semantically related data values that CFDs aim to capture.

Closer to CFDs is the tableau representation of dependencies [Wijsen 2005]. This work represents full dependencies by tableaux that also allow data values. It differs from this work in that it focuses on condensed representations of repairs and consistent query answers. It does not address the issues of context patterns, inference system, or effective techniques for detecting inconsistencies.

Beyond dependencies, error detection has been studied for cleaning census/survey data [Fellegi and Holt 1976; Garfinkel et al. 1986; Winkler 2004; 1997; Bruni and Sassano 2001]. In that setting, data inconsistencies are specified in terms of a set of editing rules. As opposed to FDs and CFDs, the editing rules do not detect inconsistencies among different tuples; instead, they aim to catch discrepancies and errors in a single tuple. The intractability of the editing-based census data repair problem is established in [Fellegi and Holt 1976], which also provides a heuristic repairing algorithm. Various improved algorithms have been developed [Garfinkel et al. 1986; Winkler 2004; 1997; Bruni and Sassano 2001], which adopt, among other things, a technique analogous to computing minimal cover of CFDs that is to "localize" errors. Statistical and mining methods have also been used for detecting inconsistencies, *e.g.,* outlier detection (see [Maletic and Marcus 2000] for a survey).

A host of work on data cleaning has focused on the merge/purge problem, also known as record linkage, entity reconciliation or duplicate removal [Fellegi and Sunter 1969; Galhardas et al. 2000; Hernandez and Stolfo 1998; Monge 2000; Winkler 1994]. That is the problem of linking pairs of records that refer to the same real-world entity in different data sets. It is commonly applied to household information in census data, mailing lists or medical records as well as many other uses.

Most work in this line of research has focused on clustering methods for grouping similar records, and is different from this work. On the other hand, there are in fact connections between record linkage and CFDs. Indeed, duplicate records are typically identified by a set of equational axioms. An example axiom is that in the US, two people with the same SSN but different names should be identified as the same person and thus should be assigned the same "key". Certain equational axioms can actually be expressed as CFDs; *e.g.,* the CFD (person: country, SSN $\rightarrow$ key, (01, _ || _)) specifies the example axiom given above. An interesting topic for future research is to extend CFDs to express equational axioms across different relations, and to explore applications of such CFDs to record linkage.

Data cleaning systems reported in the research literature include the AJAX system [Galhardas et al. 2001], which provides users with a declarative language for specifying data cleaning programs, and the Potter's Wheel system [Raman and Hellerstein 2001] that extracts structure for attribute values and uses these to flag discrepancies in the data. Most commercial ETL tools for data warehouses have little built-in data cleaning capabilities, covering mainly data transformation needs such as data type conversions, string functions, etc. [Rahm and Do 2000] presents a comprehensive survey of commercial data cleaning tools, as well as a taxonomy of current approaches to data cleaning. While a constraint repair facility will logically become part of the cleaning process supported by these systems, we are not aware of analogous functionality currently in any of the systems mentioned.

The work in [Bohannon et al. 2005] is complimentary to ours: it focuses on repairing inconsistencies based on standard FDs and inclusion dependencies, *i.e.,* to edit the instance via minimal value modification such that the updated instance satisfies the constraints, while our work focuses on static analyses of CFDs and techniques for detecting inconsistencies. Since CFDs are more expressive than FDs, not all of the detected CFD violations can be repaired by the algorithm of [Bohannon et al. 2005]. A recent extension of [Bohannon et al. 2005] studies repairing inconsistent databases based on CFDs [Cong et al. 2007]. Compared to the results of [Bohannon et al. 2005], it has been demonstrated in [Cong et al. 2007] that CFDs are more effective than traditional FDs in identifying and repairing real-life inconsistent data.

Related work on extensions of FDs. Functional dependencies were first introduced by Codd [Codd 1972] while their axiomatization is due to Armstrong [Armstrong 1974]. The implication of functional dependencies is studied both in [Beeri and Bernstein 1979] and [Maier 1980] where in the latter it is shown that the minimal cover for a set of functional dependencies can be computed in polynomial time. Table I compares the main results for static analyses of CFDs with their FD counterparts, namely, the consistency and implication analyses in general, with a predefined schema (fixed R) and without attributes with a finite domain ($|\mathsf{finattr}(R)| = 0$), as well as finite axiomatizability. Here R is a relation schema, Σ is a set of CFDs (resp. FDs), and φ is a single CFD (resp. FD) in its normal form ($R : X \rightarrow A, t_p$) (resp. $R : X \rightarrow A$), where X is a set of attributes and A is a single attribute.

A variety of extensions to FDs have been proposed for specifying constraint databases and constraint logic programs [Baudinet et al. 1999; Bra and Paredaens 1983; Maher 1997; Maher and Srivastava 1996]. Constraints of [Bra and Paredaens

Problem	CFDs	FDs
Consistency	NP-complete (Th. 3.2)	$O(1)$
Consistency (fixed R)	$O(\|\Sigma\|^2)$ (Prop. 3.6)	$O(1)$
Consistency ($\|\mathsf{finattr}(R)\| = 0$)	$O(\|\Sigma\|^2 \|\mathsf{attr}(\Sigma)\|)$ (Prop. 3.5)	$O(1)$
Implication	coNP-complete (Th. 4.3)	$O(\|\Sigma\|)$
Implication (fixed R)	$O(\|\Sigma\|^2)$ (Cor. 4.4)	$O(\|\Sigma\|)$
Implication ($\|\mathsf{finattr}(R)\| = 0$)	$O(\|\Sigma\|^2 \|\mathsf{attr}(\Sigma)\|)$ (Cor. 4.4)	$O(\|\Sigma\|)$
Finite Axiomatizability	yes (Th. 4.2)	yes

Table I. Complexity bounds for static analyses of CFDs vs. their FD counterparts

1983], also referred to as conditional functional dependencies, are of the form $(X \to Y) \to (Z \to W)$, where $X \to Y$ and $Z \to W$ are standard FDs. Constrained dependencies of [Maher 1997] extend [Bra and Paredaens 1983] by allowing $\xi \to (Z \to W)$, where ξ is an arbitrary constraint that is not necessarily an FD. In a nutshell, these dependencies are "conditional" since they are to apply the FD $Z \to W$ only to the subset of a relation that satisfies $X \to Y$ or ξ. These dependencies cannot express CFDs since $Z \to W$ does not allow patterns with constants as found in CFDs. As a result, consistency is *not* an issue for such constraints, as indicated in Section 3. For constrained dependencies with the independence of negative constraints property (*i.e.,* when for any ξ and ξ_i's, $\xi \to \xi_1 \vee \ldots \vee \xi_k \equiv \xi \to \xi_i$ for some i), a sound and complete inference system is developed for their implication analysis; in addition, the implication problem is shown to be in PTIME for constrained dependencies with this property, and coNP-hard for those without this property. The results of [Bra and Paredaens 1983; Maher 1997] are not applicable to CFDs and the proofs presented there are quite different from their CFD counterparts.

More expressive are constraint-generating dependencies (CGDs) of [Baudinet et al. 1999] and constrained tuple-generating dependencies (CTGDs) of [Maher and Srivastava 1996], both subsuming CFDs. A CGD is of the form $\forall \bar{x}(R_1(\bar{x}) \wedge \ldots \wedge R_k(\bar{x}) \wedge \xi(\bar{x}) \to \xi'(\bar{x}))$, where R_i's are relation symbols, and ξ, ξ' are arbitrary constraints that may allow constants. As noted in [Baudinet et al. 1999], it is necessary to study the consistency problem for CGDs. When ξ and ξ' are conjunctions or disjunctions of atomic formulas defined in terms of '$=, \neq$' or '$<, \leq$', it is shown that the implication problem for CGDs is already coNP-complete even when all involved attributes have an infinite domain, and that the consistency problem is NP-complete. Compared to this work, [Baudinet et al. 1999] presents stronger results for the upper bounds for the implication and consistency analyses, while this work gives stronger lower bounds. Indeed, the proofs for the lower bounds of [Baudinet et al. 1999] make use of disjunctions of atomic formulas and inequality, and thus do not work for CFDs. Finite axiomatizability and techniques for (incrementally) detecting inconsistencies are not studied in [Baudinet et al. 1999]. A CTGD is of the form $\forall \bar{x}(R_1(\bar{x}) \wedge \ldots R_k(\bar{x}) \wedge \xi \to \exists \bar{y}(R'_1(\bar{x}, \bar{y}) \wedge \ldots \wedge R'_s(\bar{x}, \bar{y}) \wedge \xi'(\bar{x}, \bar{y}))$. Since CTGDs subsume full-fledged TGDs, their implication analysis is already undecidable even in the absence of constants and ξ, ξ'. Chase procedures are presented in [Maher and Srivastava 1996] for the implication analysis of CTGDs, but the consistency problem, inference rules, inconsistency detection techniques are not considered there. While CGDs and CTGDs can express CFDs, the increased expressive power comes with the price of a higher complexity. Moreover, we are not aware of any applications of

these constraints in data cleaning.

A class of instance-level FDs (ILFDs) is studied in [Lim et al. 1996]. ILFDs are a special case of CFDs of the form $(X \rightarrow Y,\ T_p)$, where T_p contains a single tuple consisting of only constants. ILFDs are used in *entity identification*: given an entity (tuple) E in a relation R, a set of ILFDs is used to compute the values of an extended key (identifier) for E. Extended keys are also computed, through ILFDs, for each entity E' in a second relation S. If the extended keys of E and E' are equivalent, then E is considered identical to E'. Analyses of consistency and implication and inconsistency detection are not considered for ILFDs [Lim et al. 1996].

Tableau representations of dependencies. Pattern tableaux are used in *template dependencies* (TDs) [Sadri and Ullman 1982], their generalization of *tuple-generating dependencies* (TGDs) [Beeri and Vardi 1984], and *equality-generating dependencies* (EGDs) [Sadri 1980; Beeri and Vardi 1984] (See also [Maier 1983] for an historical account of these formalisms). A TD is of the form (T, w), where all pattern tuples in T and w consist of named variables only. An instance I satisfies (T, w) if for all valuations ρ such that $\rho(T) \subseteq I$, ρ can be extended such that $\rho(w) \in I$. When w is a tableau T' itself, one gets TGDs. Again neither T nor T' contains constants. It is clear that TGDs (and therefore also TDs) cannot express FDs. Moreover, the absence of constants prevents it also from expressing CFDs with a constant RHS. Conversely, TDs and TGDs cannot be expressed by CFDs. On the other hand, an EGD is of the form $(T, x = y)$ where, as before, T consists of a pattern tuples consisting of named variables only. An instance I satisfies $(T, x = y)$ if for every valuation ρ such that $\rho(T) \subseteq I$ it is the case that $\rho(x) = \rho(y)$. Although FDs can be expressed by EGDs, the absence of constants in the pattern tuples prevents it form expressing CFDs. Conversely, not all generalized FDs can be expressed by CFDs either. More recently [Wijsen 2005] studied EGDs and full TGDs that allow constants in the pattern tuples. As a consequence, these extensions can express CFDs with a variable and constant RHS, respectively. However, as mentioned above, the focus of [Wijsen 2005] is different from ours.

In the context of incomplete information [Imieliński and Lipski Jr 1984; Grahne 1991] one finds pattern tableaux in the form of Codd tables, v-tables and conditional (c-) tables. The most expressive formalism is that of c-tables. A c-table is of the form $(T, \Phi_T, \{\varphi_t \mid t \in T\})$, where T is a set of pattern tuples consisting of named variables and constants, Φ_T is a global condition defined as a conjunction of atomic equality and inequality constraints on the variables and constants, and φ_t is a condition that only applies to individual pattern tuple t in T. Now, every valuation of T that satisfies the conditions results in a different relation instance. That is, every pattern tuple is mapped onto a single instance tuple, and the cardinality of the instance is bounded by the number of pattern tuples in T. In short, each of these table formalisms is used as a representation of a (possibly infinite) set of relation instances, one instance for each valuation of the variables in the table. No instance represented by these table formalisms can include two tuples that result from two different valuations of the same pattern tuple. In contrast, all pattern tuples in the pattern tableau in a CFD are *constraints* on a *single* relation instance. This single instance can contain any number of tuples that are all instantiations of the same pattern tuple. Hence, the use of tableaux in the context of incomplete

information entirely differs from pattern tableaux in CFDs.

8. CONCLUSIONS

We have introduced CFDs as an extension of FDs, and shown that CFDs are capable of capturing inconsistencies beyond what traditional FDs can detect. We have provided complexity results and techniques for reasoning about CFDs. More specifically, we have shown that the consistency problem for CFDs is NP-complete, as opposed to the trivial consistency analysis of traditional FDs, and that the implication problem for CFDs is coNP-complete, in contrast to the linear-time complexity of their traditional counterpart. In practical data-cleaning settings, the relational schema is often predefined, and these problems become decidable in low PTIME. Furthermore, we have developed an approximation-factor preserving reduction from the consistency problem for CFDs to MAXGSAT, and a sound and complete inference system for the implication analysis of CFDs. For applications of CFDs in data cleaning, we have proposed (incremental) SQL-based techniques for detecting violations of CFDs. We have also experimentally evaluated our detection techniques. These results establish a constraint theory for CFDs and are also promising in developing a practical constraint-based tool for data cleaning.

There is naturally much more to be done. First, to clean data, constraints beyond CFDs are certainly needed. There has been recent work on conditional inclusion dependencies, denoted by CINDs, which are defined along the same lines as CFDs and are demonstrated useful in data cleaning and schema matching [Bravo et al. 2007]. The static analysis of these conditional dependencies becomes, however, more intriguing. In particular, the consistency and implication problems for CFDs and CINDs together become undecidable. To cope with this it is necessary to find effective and efficient heuristic algorithms for the consistency and implication analyses of these conditional constraints. Second, automated methods for discovering CFDs and CINDs are certainly an interesting topic. It is nontrivial to identify all sensible pattern tuples without over-populating pattern tableaux. Third, another important issue concerns how to effectively remove inconsistencies from data after inconsistencies are detected in a database I based on a set Σ of conditional constraints. This, referred to as constraint repair [Arenas et al. 2003], aims to find a database that satisfies Σ and minimally differs from I. While there has been recent preliminary work on this issue based on CFDs alone [Cong et al. 2007], it deserves further investigation in the presence of both CFDs and CINDs.

REFERENCES

Abiteboul, S., Hull, R., and Vianu, V. 1995. *Foundations of Databases.* Addison-Wesley.

Arenas, M., Bertossi, L. E., and Chomicki, J. 2003. Consistent query answers in inconsistent databases. *Theory and Practice of Logic Programming 3,* 4-5, 393–424.

Armstrong, W. W. 1974. Dependency structures of data base relationships. In *Proc. IFIP World Computer Congress.* 580–583.

Baudinet, M., Chomicki, J., and Wolper, P. 1999. Constraint-Generating Dependencies. *J. Comput. Syst. Sci. (JCSS) 59,* 1, 94–115.

Beeri, C. and Bernstein, P. A. 1979. Computational problems related to the design of normal form relational schemas. *ACM Trans. on Database Systems 4,* 1, 30–59.

Beeri, C. and Vardi, M. 1984. A proof procedure for data dependencies. *J. ACM 31,* 4, 718–741.

Bertossi, L. and Chomicki, J. 2003. Query answering in inconsistent databases. In *Logics for Emerging Applications of Databases*. 43–83.

Bohannon, P., Fan, W., Flaster, M., and Rastogi, R. 2005. A cost-based model and effective heuristic for repairing constraints by value modification. In *Proc. Int'l Conf. on Management of Data (SIGMOD)*. 143–154.

Bohannon, P., Fan, W., Geerts, F., Jia, X., and Kementsietsidis, A. 2007. Conditional functional dependencies for data cleaning. In *Proc. Int'l Conf. on Data Engineering (ICDE)*. 746–755.

Bra, P. D. and Paredaens, J. 1983. Conditional dependencies for horizontal decompositions. In *Colloquium on Automata, Languages and Programming*. 67–82.

Bravo, L. and Bertossi, L. 2003. Logic programs for consistently querying data integration systems. In *Proc. Int'l Joint Conf. on Artificial Intelligence*. 10–15.

Bravo, L., Fan, W., and Ma, S. 2007. Extending dependencies with conditions. In *Proc. Int'l Conf. on Very Large Databases (VLDB)*. 243–254.

Bruni, R. and Sassano, A. 2001. Errors detection and correction in large scale data collecting. In *Proc. Int'l Conf. on Advances in Intelligent Data Analysis (IDA)*. 84–94.

Cali, A., Lembo, D., and Rosati, R. 2003a. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *Proc. Symp. on Principles of Database Systems (PODS)*. 260–271.

Cali, A., Lembo, D., and Rosati, R. 2003b. Query rewriting and answering under constraints in data integration systems. In *Proc. Int'l Joint Conf. on Artificial Intelligence*. 16–21.

Chomicki, J. and Marcinkowski, J. 2005a. Minimal-change integrity maintenance using tuple deletions. *Information and Computation 197,* 1-2, 90–121.

Chomicki, J. and Marcinkowski, J. 2005b. On the computational complexity of minimal-change integrity maintenance in relational databases. In *Inconsistency Tolerance*. 119–150.

Codd, E. F. 1972. Relational completeness of data base sublanguages. In *Data Base Systems: Courant Computer Science Symposia Series 6*. Prentice-Hall, 65–98.

Cong, G., Fan, W., Geerts, F., Jia, X., and Ma, S. 2007. Improving data quality: Consistency and accuracy. In *Proc. Int'l Conf. on Very Large Databases (VLDB)*. 315–326.

Eckerson, W. W. 2002. Data Quality and the Bottom Line: Achieving Business Success through a Commitment to High Quality Data. Tech. rep., The Data Warehousing Institute. *http://www.tdwi.org/research/display.aspx?ID=6064*.

Fellegi, I. and Holt, D. 1976. A systematic approach to automatic edit and imputation. *J. of the American Statistical Association 71,* 353, 17–35.

Fellegi, I. P. and Sunter, A. B. 1969. A theory for record linkage. *J. of the American Statistical Association 64,* 328, 1183–1210.

Franconi, E., Palma, A. L., Leone, N., Perri, S., and Scarcello, F. 2001. Census data repair: a challenging application of disjunctive logic programming. In *Proc. Artificial Intelligence on Logic for Programming (LPAR)*. 561–578.

Galhardas, H., Florescu, D., Shasha, D., and Simon, E. 2000. AJAX: An extensible data cleaning tool. In *Proc. Int'l Conf. on Management of Data (SIGMOD)*. 590.

Galhardas, H., Florescu, D., Shasha, D., Simon, E., and Saita, C.-A. 2001. Declarative data cleaning: Language, model and algorithms. In *Proc. Int'l Conf. on Very Large Databases (VLDB)*. 371–380.

Garey, M. and Johnson, D. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company.

Garfinkel, R. S., Kunnathur, A. S., and Liepins, G. E. 1986. Optimal imputation of erroneous data: Categorical data, general edits. *Operational Research 34,* 5, 744–751.

Gertz, M. and Lipeck, U. 1995. A diagnostic approach to repairing constraint violations in databases. In *Int'l Workshop on Principles of Diagnosis (DX)*. 65–72.

Grahne, G. 1991. *The Problem of Incomplete Information in Relational Databases*. Springer.

Greco, G., Greco, S., and Zumpano, E. 2003. A logical framework for querying and repairing inconsistent databases. *IEEE Trans. Knowl. Data Eng 15,* 6, 1389–1408.

Hernandez, M. A. and Stolfo, S. 1998. "Real-World Data is Dirty: Data Cleansing and the Merge/Purge Problem". *Data Mining and Knowledge Discovery 2,* 1, 9–37.

Imieliński, T. and Lipski Jr, W. 1984. Incomplete information in relational databases. *J. of the ACM 31,* 4, 761–791.

Lim, E.-P., Srivastava, J., Prabhakar, S., and Richardson, J. 1996. Entity identification in database integration. *Inf. Sci. 89,* 1-2, 1–38.

Maher, M. J. 1997. Constrained dependencies. *Theoretical Computer Science 173,* 1, 113–149.

Maher, M. J. and Srivastava, D. 1996. Chasing Constrained Tuple-Generating Dependencies. In *Proc. Symp. on Principles of Database Systems (PODS)*. 128–138.

Maier, D. 1980. Minimum covers in relational database model. *J. of the ACM 27,* 4, 664–674.

Maier, D. 1983. *The Theory of Relational Databases*. Computer Science Press.

Maletic, J. I. and Marcus, A. 2000. Data cleansing: Beyond integrity analysis. In *Proc. Conf. on Information Quality (IQ)*. 200–209.

Monge, A. E. 2000. Matching algorithms within a duplicate detection system. *IEEE Data Eng. Bull. 23,* 4, 14–20.

Papadimitriou, C. H. 1994. *Computational Complexity*. Addison Wesley.

Rahm, E. and Do, H. H. 2000. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull. 23,* 4, 3–13.

Raman, V. and Hellerstein, J. M. 2001. Potter's wheel: An interactive data cleaning system. In *Proc. Int'l Conf. on Very Large Databases (VLDB)*. 381–390.

Sadri, F. 1980. Data dependencies in the relational model of data: A generalization. PhD thesis, Princeton University.

Sadri, F. and Ullman, J. 1982. Template dependencies: A large class of dependencies in relational databases and its complete axiomatization. *J. ACM 29,* 2, 363–372.

Shilakes, C. C. and Tylman, J. 1998. Enterprise information portals. Tech. rep., Merrill Lynch, Inc., New York, NY. Nov.

Vazirani, V. V. 2003. *Approximation Algorithms*. Springer.

Wijsen, J. 2005. Database repairing using updates. *ACM Trans. on Database Systems 30,* 3, 722–768.

Winkler, W. E. 1994. Advanced methods for record linkage. Tech. rep., Statistical Research Division, U.S. Bureau of the Census.

Winkler, W. E. 1997. Set-covering and editing discrete data. In *Proc. of the Section on survey research methods, American statistical association.* 564–569.

Winkler, W. E. 2004. Methods for evaluating and creating data quality. *Information Systems 29,* 7, 531–550.